Kevin Townsend

LEARNING TO USE THE TI99/4A COMPUTER

Learning to Use the TI 99/4A Computer

Learning to Use the TI 99/4A Computer

by Kevin Townsend

Gower

Published by
Gower Publishing Company Limited,
Gower House, Croft Road, Aldershot,
Hampshire GU11 3HR, England

British Library Cataloguing in Publication Data

Townsend, Kevin
Learning to Use the TI 99/4A Computer.—(Learning to use)
1. TI 99/4A (Computer)
I. Title
001.64'04 QA76.8.T

ISBN 0 566 03492 1

Typeset and printed in Great Britain
at FD Graphics, Fleet, Hants.

Contents

List of Figures

Foreword

This series of books is designed to fill the gap left by nearly every other book ever published on microcomputing. For some reason there has always been a surfeit of books which assume a thorough basic knowledge; but few books to give you that knowledge in the first place. Even the manuals supplied with most systems seem to assume you understand the main principles from the beginning: in fact, most are only useful if you don't need them! This open-ended series of books from the Gower Publishing Company provides the new computer user with a jargon-free introduction to his machine. He doesn't need to have any previous computing experience, for the book itself provides all the relevant information necessary to get started: and if he does come across any strange new jargon, there is a useful glossary at the back. It is not, and is not intended to be, a text book on BASIC programming; but after reading this book even the complete beginner will be able to refer to a text book with complete confidence.

Most young readers, and the books are primarily (but not solely!) designed for young readers, will be using their machines at school or at home. It was probably bought by parents to help the family get used to the new technology; or perhaps it is the school's new system to help in teaching computer science. If it is used at home, it is almost certain that parents will join in the learning process – and in many cases, they may well start to monopolise it! Fifty years ago, these same parents would have bought their children a train set, and would then play with it for hours on end themselves. We can expect the same to happen with the new micros, for programming can become a very addictive pastime! This series is almost designed as much for the adult novice as it is for the younger reader.

Computing and computer skills are a modern concept, and we will all need to understand at least the basic principles in a world that is becoming increasingly computer orientated. Since our future is to be based on the computer as a useful tool to help run our factories, offices, schools and homes, we need to know how to get the most out of them. At best, the computer can be a useful, profitable, enjoyable, and beneficial aid; and at worst it can be an expensive, useless conglomeration of electronic junk. The *Learning to Use* series of books is designed to help new users get the best from their computers as quickly and as easily as possible.

The books describe a number of applications for each computer, including business, education and hobbyist. Furthermore, a simple and direct introduction to programming is included in such a way as to motivate further investigation of the computer and its capabilities. Each computer's ability to draw pictures and diagrams, in black and white or colour, is explored and explained, and programs for a large number of graphics applications are presented. Wherever available, details of each system's sound reproducing capabilities, including example programs and a version of the national anthem, are also included. For the programs, the series is indebted to William Turner, a lecturer in statistics at the Oxford Polytechnic – and no mean programmer! – who has had the unenviable task of converting somebody else's programs for each new book. The Tower of Hanoi game at the end of each book is entirely his own program and illustrates the teaching philosophy he employs at his college: elegant simplicity.

Two further commendations are necessary: to Michael Fluskey of Newtech who first conceived the idea for this series and has throughout been the driving force behind it; and to Garry Marshall who wrote the first book in the series: *Learning to Use the PET Computer*. It was the joint work of these two that developed the basic structure that is now used throughout the series.

This said, I can only wish the reader as much enjoyment from the book and his computer, as I have from mine.

Kevin Townsend
Editor, Micro Software and Systems Magazine

Chapter 1

Introduction to the Texas Instruments 99/4A

What is the Texas Instruments 99/4A?

The Texas Instruments 99/4A (which, for the rest of this book, we shall often abbreviate to 'the TI 99/4A') is a computer. It is usually called a microcomputer (and sometimes a personal computer or home computer) because it is extremely small compared to early computers – and also because its electronic 'heart' is a microprocessor. As you can see from Figure 1.1, the TI 99/4A appears like a case with a keyboard, rather like a conventional typewriter. In use, it requires a screen of some sort for its display. In order to keep down costs, it has been designed to work primarily with an ordinary television set (either black and white, or colour). Inside the TI 99/4A there are a number of integrated circuits, or chips as shown in Figure 1.2. One contains the microprocessor. Others

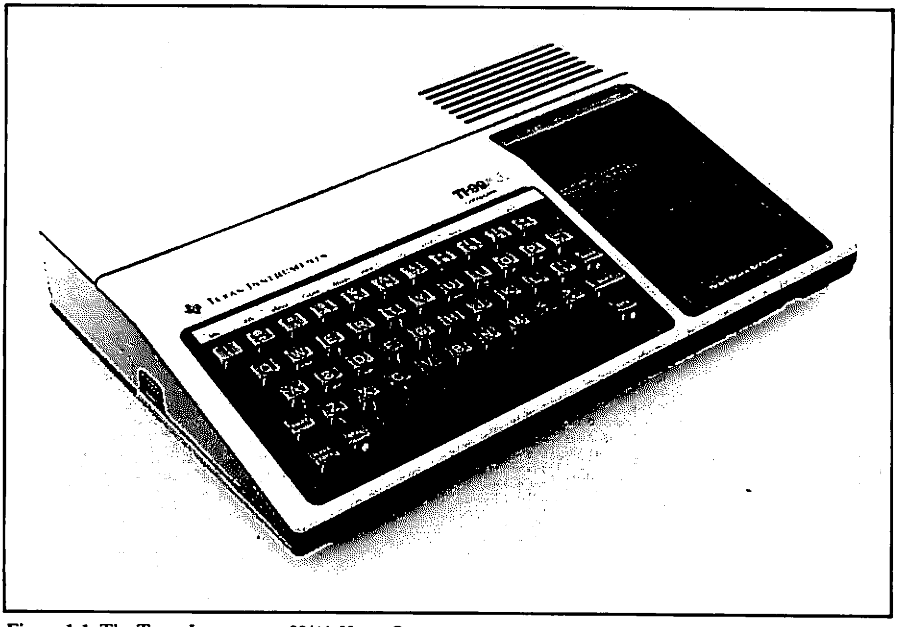


Figure 1.1 The Texas Instruments 99/4A Home Computer.

provide the computer's memory, and can store information. Initially, there is no need to worry about the inside of the computer. The electronic circuitry and the devices that make the TI 99/4A work are fascinating, but a detailed understanding of them is certainly not necessary in order to use the computer – and this introductory book is about learning to use the TI 99/4A computer.

The main feature, the keyboard, is for communicating to the computer. Commands that are to be obeyed, and information that is to be stored, can simply be typed in. Because the keyboard is set out in the same way as a typewriter, a good typist can type almost as quickly as on an ordinary typewriter. Notice, however, that if you try to type too fast, and do not perhaps press the keys firmly enough, it is quite possible for the TI 99/4A to miss some of the keys you think you have pressed. Nevertheless, it is a good idea from the start to try to use the professional 'five-finger' typing techniques rather than one-finger tapping: in the long run, this will save a considerable amount of time. Once the correct connections are made to the television set, anything you type on the keyboard will automatically appear on the screen.

The TI 99/4A computer possesses a number of what are called 'screen editing facilities'. These make it fairly simple and easy for you to 'edit' your typing, ie to correct errors, to make changes, and to arrange for the revised typing to appear on the screen. The TI 99/4A's screen editing facilities have been very carefully thought out. With a little practice, you will find them easy to use.

Besides letters and numbers, the TI 99/4A also lets you produce simple pictures on the screen. This facility is known as 'graphics', and is an impressive bonus to the use of the computer. The imaginative use of pictures, diagrams and graphs enlivens the presentation of information, and can be used in computer games, in business applications, and in educational programs.

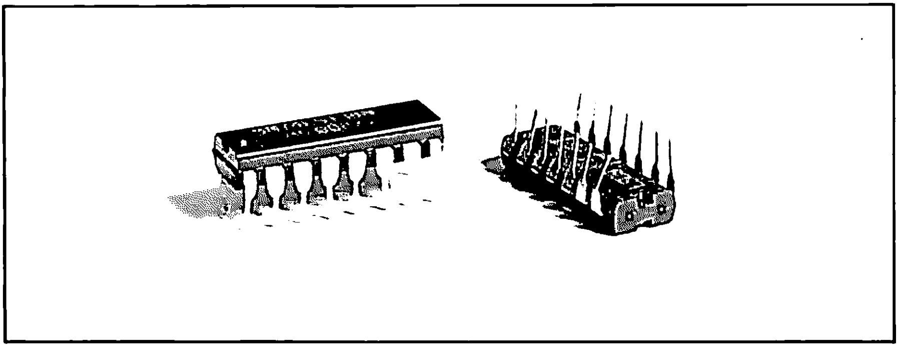


Figure 1.2 A large scale integrated circuit: chip.

The TI 99/4A is light and compact, and is small enough to be carried from room to room, or from house to house, or even from schoolroom to schoolroom. It can be set up and used in its new location quickly and easily, for it needs only to be plugged into the mains and connected to an ordinary domestic television set. Furthermore, as soon as it is switched on it is ready to accept commands typed in at the keyboard, provided only that they are typed in a language that the computer understands. The language is BASIC, and it enables you to issue commands which are promptly and automatically obeyed by the TI 99/4A computer. There are many different versions of BASIC. The TI 99/4A has two options: TI BASIC, which is held inside the computer; and TI Extended BASIC, which is supplied on a plug-in command module. Throughout this book, we shall concentrate on the inbuilt TI BASIC that is supplied automatically with the system.

How was the TI 99/4A developed?

The two main events that have hastened the advance of microelectronics, and microprocessors in particular, are the space race of the 1960s and the general requirements of the world's various defence organisations (particularly the US Department of Defense, and the UK Ministry of Defence) ever since the end of the Second World War. In the space race, because the US rockets were less powerful than those of the Russians, the Americans needed to reduce the size and weight of everything that had to be carried by their rockets – including the electronics. At the same time, defence leaders began to demand computers that were small enough, light enough, and tough enough to be carried around in, and survive, battle conditions. In particular, this stimulated the American electronics industry to investigate and develop means of miniaturising electronic circuitry – and the result is the microprocessor, often referred to as the chip. Not only is this extremely small (smaller than the size of the average finger-nail) and relatively powerful (as powerful as the early computers that would fill an average sized room), it is a multi-purpose device that can perform any electronic function for which it can be programmed. This versatility has led to the use of microprocessors in a wide and ever-growing range of applications (the most common, and best known, is, of course, as the 'heart' of a general purpose microcomputer like the TI 99/4A). The consequent mass production of microprocessors has caused the cost per unit (of the microprocessor – not the microcomputer!) to drop to just a few pence.

The story of Texas Instruments and the TI 99/4A in particular goes back some 50 years. It started on May 16th, 1930 when two young scientists, John Clarence Karcher and Eugene McDermott, signed a charter for Geophysical Service Inc. Some years earlier Karcher had had the idea that it would be possible to calculate the depth of geological strata by bouncing sound waves off them. He believed that this technique, now known as Reflection Seismology, could be used to determine the shape, depth and nature of subsurface structures and thereby to indicate the possibility of oil being present in them.

Karcher's theory proved to be workable in practice and within a year the company, based in a small office in downtown Dallas, was operating in Texas, in California and in Mexico. It was a period of steady growth, with continuous developments of equipment and technique, which lasted until the outbreak of World War II.

By 1951 the manufacturing and exploration activities had both grown so much that it was decided to set them up as separate corporate structures. The former became Texas Instruments Inc – with GSI, the exploration company, as a wholly-owned subsidiary.

Since those earliest days, Texas Instruments has been a major user of commercial computers. One might say that TI and the computer industry grew up together. While the computer was in its infancy TI designed and built its own special purpose machines for the processing of seismic data and, in 1971 produced the ASC – an advanced scientific computer which, at that time, was the largest in the world.

Texas Instruments entered the commercial minicomputer market in the late '60s. Since that time it has been actively marketing data processing equipment in Europe and now serves a wide spectrum of commerce and industry from over 40 West European locations.

In March 1967 Texas Instruments engineers, Jack Kilby, Jerry Merryman and James Van Tassel completed the world's first electronic, handheld calculator. This miniature calculator is now in the permanent collection of the Smithsonian Institute, Washington DC. The patent for the invention was assigned to TI on June 25th 1974.

In 1969 TI entered into a series of contracts for integrated circuits to be used in calculators which would be produced by other manufacturers. Then, in 1972, TI formed its own calculator division selling its first product, the TD datamath. Since then Texas Instruments has continually introduced new calculator ranges and is now one of the world's leading manufacturers.

TI is also undoubtedly the world leader in electronic educational products. The first of these was the Little Professor, introduced in 1976. It looked like a calculator but it posed mathematical questions rather than giving the answers. The philosophy is very simple: by making the learning process fun for a child, the child will want to keep on learning. In 1978 TI incorporated its newest invention – the single chip speech synthesizer –

into the Speak & Spell learning aid, one of the most innovative consumer products of all time.

With so much experience in the manufacture of semiconductor components and computer items for the professional market plus its commitment to electronic products for the consumer market, it was probably inevitable that Texas Instruments should, at some time, produce a home computer. The TI 99/4 (now superseded by the TI 99/4A) was just that – a computer designed truly for the home. The keynote is ease-of-operation even by the user who has never touched a computer in his life before. The computer is also the pioneer of Solid State Software technology which allows electronic services to be reprogrammed through the use of interchangeable plug-in modules. This technique, applied to the TI 99/4A, allows even a novice to program the computer in a moment. Simply by plugging in the appropriate command module the computer becomes an arcade game, a learning aid for children, or a household money management system – there are scores and scores of different software packages to choose from.

The TI 99/4A is based on the 9900 Family 16-bit microprocessor, a processor manufactured by Texas Instruments and similar to the one used in many of the amusement arcade games and identical to that used in a number of other successful microcomputers. Ironically, given the military origins of microelectronics, these microprocessors are more advanced examples of the technology than those used in the guidance systems of the inter-continental ballistic missiles, and even the ultra-modern Exocet-type missiles! The processor is the heart (and brain!) of the TI 99/4A. Although it does all the hard work, all the computing and calculations, you can tap its potential and make it work for you without having any knowledge of how it functions.

While many manufacturers release several versions of any new computer (usually based on the size of its RAM, or internal memory), Texas Instruments has so far released just the single machine: the TI 99/4A. At the time of writing however, a TI 99/4B appears to be imminent. The size of its memory is 16K RAM (expandable to 48K), or 16 x 1024 bytes of random access memory. One byte is actually made up of 8 bits of information, but the important thing to remember is that it takes approximately one byte to store a single character. Thus, the TI 99/4A can store up to a little over 16,000 characters in its internal memory.

When using the computer, the internal memory (comprising both RAM and ROM – see glossary) has to store both the program that is operating, the language interpreter that converts the program instructions into machine instructions, and any information you have to type in. Thus, smaller computers like the old 8K PETs, and the newer 1K ZX80s are all capable of operating simple games programs, but are probably not large enough to operate the more sophisticated fantasy games like Dungeons and Dragons; and certainly not large enough for the majority of business uses. But since the first TI 99/4A produced has a fairly substantial memory size, certainly in terms of home computers, it is more than likely that Texas Instruments has further plans for the machines, even into basic business applications.

What can the TI 99/4A do?

Fundamentally, the TI 99/4A microcomputer can do anything that you can tell it to do. That is, it will obey any instruction or set of instructions that is correctly given. A set of instructions to a computer is usually called a computer program, and is written in a special language called a programming language. Like any other computer, the TI 99/4A executes programs and does what you tell it to do. Thus, one way to make use of the computer is to learn to program in its own language, which is BASIC. Now although BASIC is the natural language of the TI 99/4A computer, it is not the natural language of the 9900 Family microprocessor. The BASIC program must therefore be translated into the language, or code (known as 'machine code'), that is understood by the microprocessor. This second level of translation happens automatically when you 'RUN' a program, and you will be unaware of it when you are using the computer. It is possible to write programs directly in the TI 99/4A's machine code, but this book will not go into the methods. Machine code programs are considerably more difficult to write than BASIC programs, even though they operate at a much faster speed. One reason for this is that there is no time lost in the translation between BASIC and machine code. Another reason is that no translation is ever as good as the original. Just as a word for word translation of Shakespeare into French can never be as good as either Shakespeare's English, or Molières French, so a translation from BASIC to machine code can never be as efficient as a program written directly in machine code.

However, it is not essential to be an expert programmer to use the TI 99/4A computer since a growing number of ready-made programs can be purchased. These programs either come on a cassette tape, from which they are transferred into the memory via an ordinary cassette recorder/ player unit; or as a games cartridge that can be plugged into a cartridge slot on the right-hand side (looking from the front) of the front of the computer. Since many programs are already available, you may like to have a look at the hobby computer magazines for their advertisements. A much wider range will soon be available. Many commercial firms will be supplying programs specifically for the TI 99/4A, and some are already advertised in the press. Texas Instruments itself produces a wide range of games and other programs. These are usually supplied on plug-in command modules and are written in machine code.

Programs that are purchased from a different source usually arrive on an ordinary cassette, or possibly a floppy disk. Figure 1.3, shows just such a cassette and a floppy disk. To transfer a program to the TI 99/4A requires either a cassette tape system (player/recorder) or a disk unit. Figure 1.4 shows a cassette tape recorder attached to the TI 99/4A. Figure 1.5 shows a command module on its own and plugged into the computer. The programs

Figure 1.3 A cassette tape and a floppy diskette.

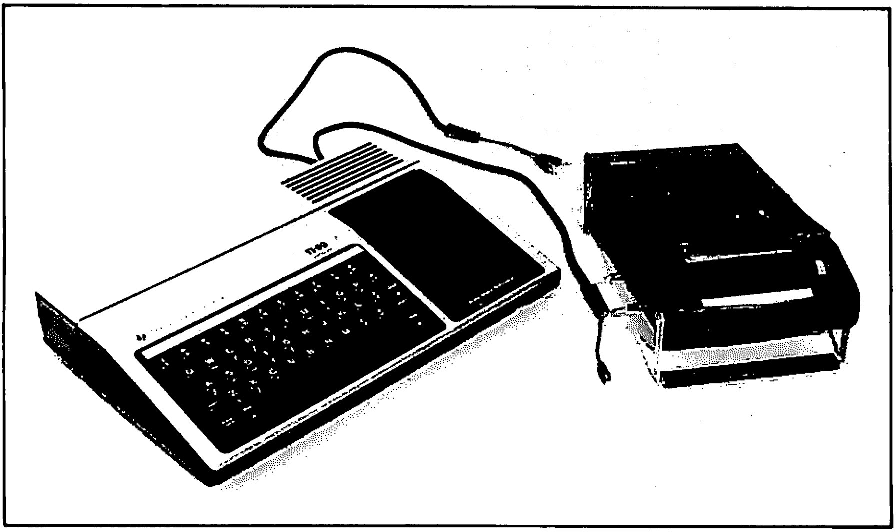


Figure 1.4 A cassette recorder attached to the TI 99/4A.

themselves are often referred to as 'software', in contrast to the computer itself, which is known as the 'hardware'.

The TI 99/4A computer can do many things. As with most microcomputers, you can often make it do these things without having any personal knowledge of programming. Nevertheless, it is often useful to be able to program, if only to amend or modify an existing program. Besides, programming is fun! It is easy to do, and it provides a means of expressing

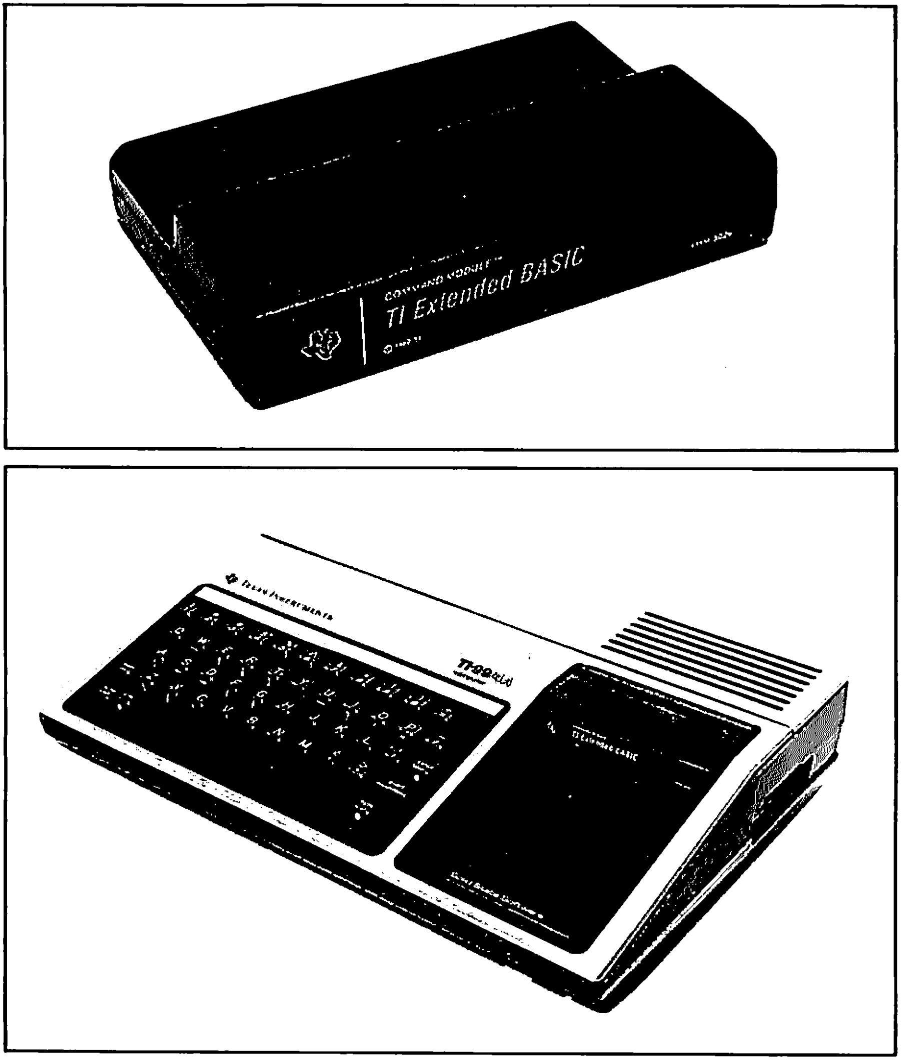


Figure 1.5 (a) A TI Solid State Software command module.
(b) A command module inserted into the computer console.

and communicating your own ideas to the computer so that it can test them for you.

How can the TI 99/4A be extended?

Besides performing computations and storing information the TI 99/4A can, again like most other computers, be used in conjunction with other devices. Units which can be connected to it, and controlled by it, are called 'peripherals'. You met one of them in the previous section: the cassette recorder. This is a peripheral device used for the permanent storage of information or programs by means of a magnetic pattern on the tape. The TI 99/4A is supplied with a cable that will connect it and the recorder. In fact, the TI 99/4A is almost unique in offering the ability to attach two tape recorders simultaneously. (Figure 1.6 shows a rear view of the computer with a number of the various peripheral connection sockets, including the cassette recorder port.) One end of the cable connects to the left-hand socket (viewed from the rear), while the other end connects to the 'earphone' and 'microphone' sockets of the recorder. A third wire of one of the cables can optionally connect to the recorder's socket usually labelled 'remote'. If your recorder has one of these sockets, the TI 99/4A can control the cassette motor, and can stop the cassette automatically as soon as it has successfully received the required information on the tape. We recommend, however, that you disregard the 'remote' wire and use the recorder manually.

For many computer applications it is useful to have the results of the computer's work in written form to provide a permanent record of the results of computations. It is also a useful aid for when you start to write your own programs, and particularly when you start to amend programs, to have a written copy. This printed output is called either a 'printout', or a 'listing'. A printout usually refers to the results of computations, while a listing refers specifically to a printed copy of a program. Obviously, a

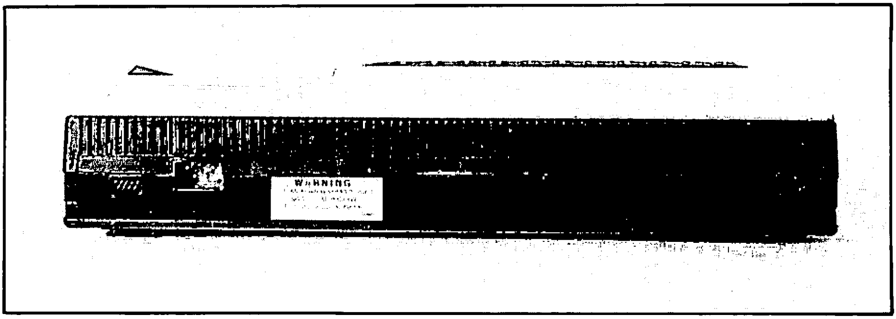


Figure 1.6 Rear view of the TI 99/4A.

computer printer is an essential peripheral for obtaining permanent printouts and listings. It can be attached to the TI 99/4A as shown in Figure 1.7. If you wish to buy a printer, ask your supplier, or Texas Instruments, for advice.

All peripherals are attached to the TI 99/4A using the connections on the sides, rear and top front of the computer. Figures 1.8, 1.9, 1.10 and 1.11 show the various views of the TI 99/4A with the different connection sockets, or ports. A 'port' is another name for a connection point between a computer and the outside world because it is similar to the way in which an airport or shipping port connects a country or region to its own outside world.

The TI 99/4A also has a device known as the Peripheral Expansion System, which allows you to add accessories to your computer system in a single, convenient location by inserting them in the peripheral system itself. The package includes the Peripheral Expansion System, and the Peripheral Expansion card with a connecting cable. The latter two combine to serve as an interface between the computer and the accessories in the unit. With this system attached to the 99/4A, you can:

1. Increase the capabilities of your computer system with a variety of accessories in the form of slide-in cards.
2. Install a TI Disk Memory Drive in the compartment designed for this purpose.

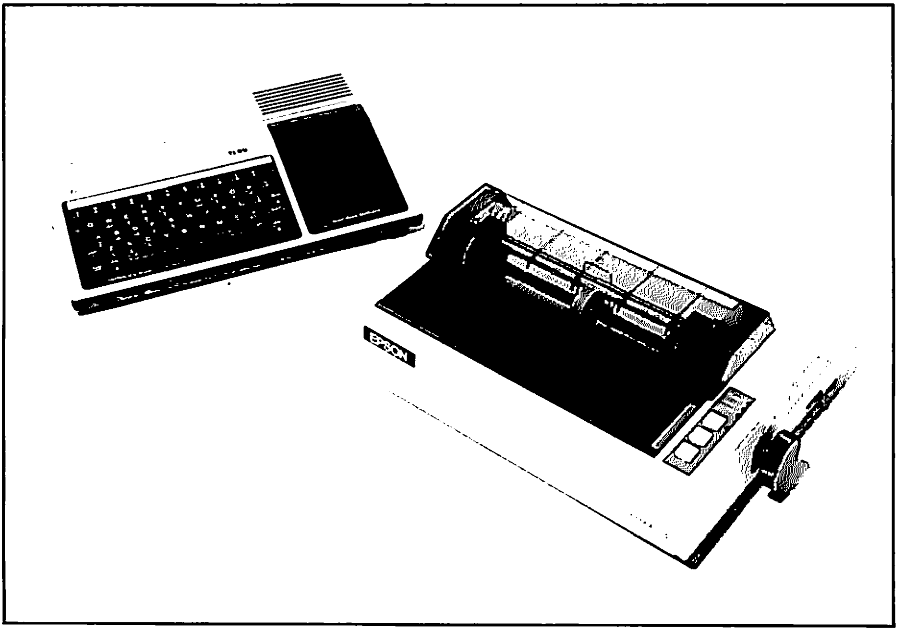


Figure 1.7 A printer attached to the TI 99/4A.

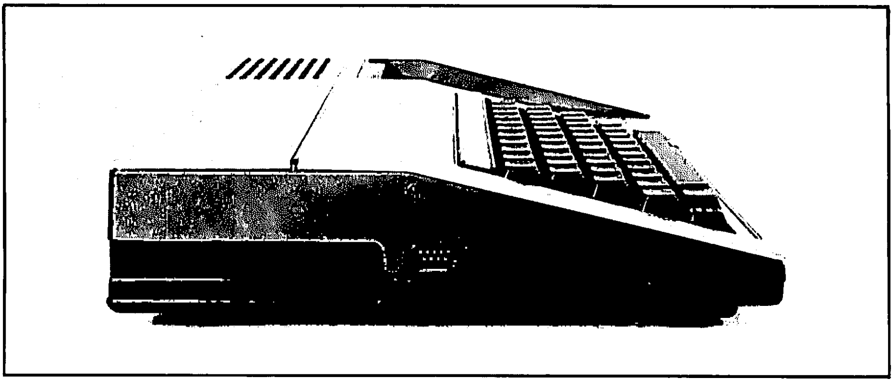


Figure 1.8 Side view: remote control port (joysticks).

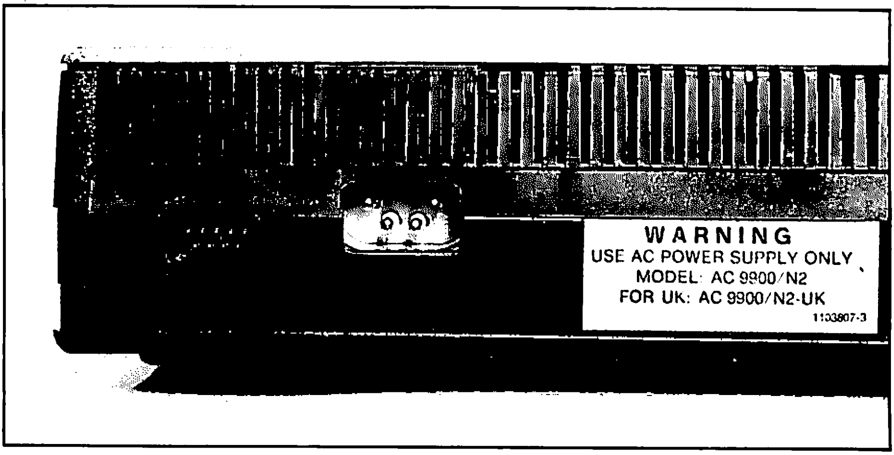


Figure 1.9 Rear view: power socket and cassette recorder port, display port.

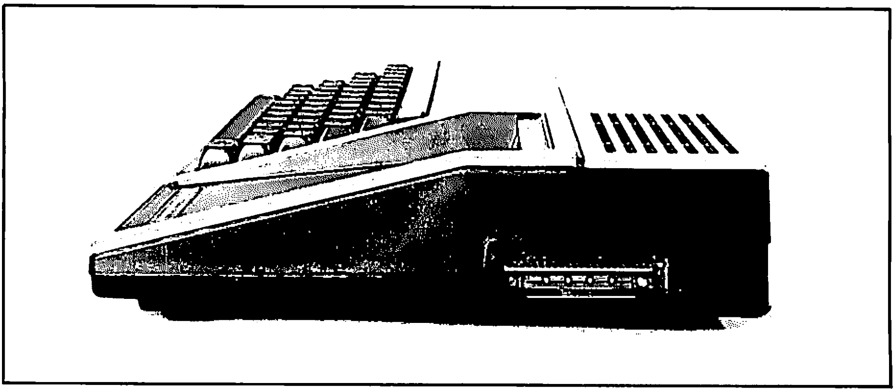


Figure 1.10 Side view: Speech Synthesizer port.

3. Connect the unit to the computer via cable in order to provide flexibility in the placement of equipment.

By removing the top of the unit and sliding the accessory cards into the slots provided, the Peripheral Expansion System can hold up to seven computer accessories, such as the TI Disk Drive Controller card, the TI RS232 Interface Card, the TI Memory Expansion Card, and the TI P-Code Peripheral Card, in addition to the Peripheral Extension Card.

What are some typical applications of the TI 99/4A?

The TI 99/4A computer was designed with many serious applications in mind. The areas in which it can be used can be broadly classified as: in the home for personal and recreational use; in educational institutions; and, to a certain (but growing) extent, in business.

For personal use, there are games programs of many kinds already available, and more becoming available all the time. There is even one, called the Tower of Hanoi (a classic ancient logic problem) at the end of this book, that you can type into the TI 99/4A and play. Using a computer for playing games is sometimes criticised as a frivolous use of an advanced electronic device, and there is no doubt that many games are lighthearted. Nevertheless, they do serve a useful purpose for relaxation and entertainment. Furthermore, there are many imaginative and stimulating games that, like the Tower of Hanoi, have a definite educational value. Other games can teach or help to develop attributes ranging from simple manipulation and co-ordination skills in children, and the mental disciplines required to find solution methods for puzzles, to testing strategies and tactics against situations presented by the computer. (The TI 99/4A is also unique among the current range of home computers in having perhaps

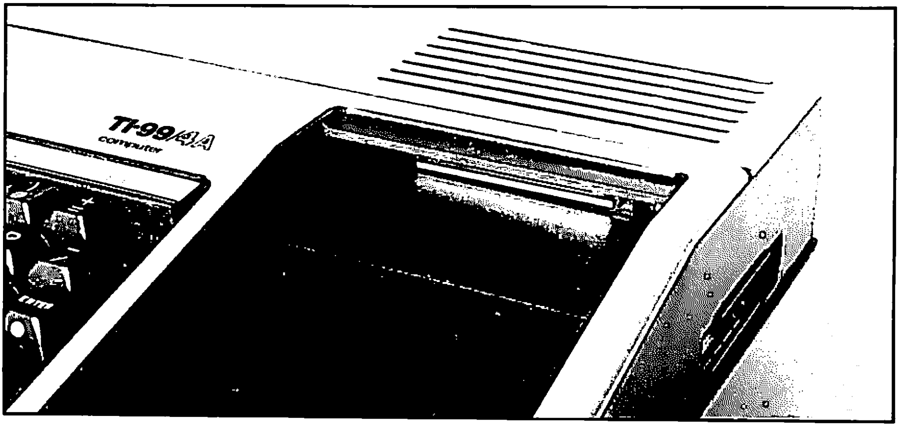


Figure 1.11 Front view: command module port.

the world's best speech synthesizer. Later on in the book is an example of a spelling test program using the synthesizer to ask you to spell certain words. There can be no doubt about the educational value of such programs, but at the same time they are sufficiently enjoyable to be almost classed as lighthearted games!) There are also many chess-playing programs of a formidable standard, and, perhaps spurred by playing such games, you might even be writing your own games programs in the near future.

The presence of a TI 99/4A in the home means that the educational actitivies need not be restricted to schools and colleges. Computer assisted learning packages have been available for some time on other computers and are already being converted for use on the TI 99/4A. These can be used just as effectively at home as at school. Computer assisted learning is not, however, intended to replace teachers, but to assist them by providing another tool. In a post-industrial society (it is often claimed that, just as the country went through an industrial revolution during the last century, so it is now going through a technological and information revolution), it is important to expose everyone, as early as possible, to the current technology. Only in this way will young people today be made aware of the possibilities presented by modern electronics, and be in a position to take advantage of the potential of computers. The presence of a TI 99/4A computer as an everyday item in the home or school can help achieve this objective.

Examples of Texas Instruments' educational programs include:

1. Division. Animation, colour and graphics are all used to teach the meaning of division and the basic facts of division. The operator is automatically provided with extra help if his performance is low.
2. Music Maker. A music composition package that lets even the novice composer create computer music by simply arranging notes on an electronic musical staff.
3. Video Chess. A powerful but easy to use package that keeps track of all the moves. You can simply use the computer as an electronic board, or you can use it as your opponent. It allows you to choose the level of difficulty, and will even save a particular game for later replay.

In schools, microcomputers have many valid uses, ranging over computer assisted learning, instructional programs, and the use of quiz programs. In higher education, the TI 99/4A is ideal for activities such as Sixth Form and undergraduate programming projects. It is particularly strong and useful in the range of alternative programming languages available, including TI Extended BASIC, UCSD Pascal, TI-LOGO and Editor/Assembler. Although these are for the more advanced user and are not examined in any depth in this particular book, it is worth noting some of the main points.

Extended TI BASIC is very similar to the TI BASIC that we shall use in this book, except that it contains extra functions and features that make advanced programming easier and more efficient.

UCSD Pascal is a system that was developed to aid the use of the Pascal programming language, a language that was itself originally developed in Switzerland as an aid to the computer learning environment. Its main feature is that it has an inbuilt degree of software portability, which means that a program written in UCSD Pascal on a different make of machine will also run on the TI 99/4A, provided only that the two hardware configurations are similar (for example, if the program was originally written on a machine with a floppy disk and a printer, it will most probably require the TI 99/4A to have such peripherals as well).

TI-LOGO is a computer language based on a philosophy of education developed by Professor Seymour Papert and the staff of the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. The main principle of LOGO is the creation of computer based environments in which formal learning (such as mathematics) can occur in a natural manner.

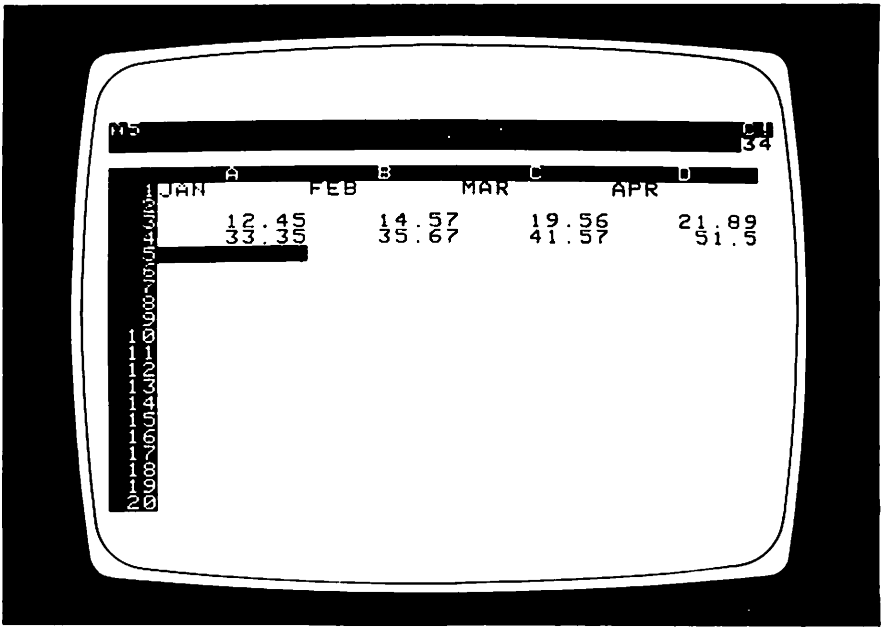
The Editor/Assembler package extends the flexibility of the TI 99/4A by allowing users to program in assembly language. It gives the programmer a direct access to all the computer's features such as sound, speech, graphics and interfaces, as well as enabling the highest possible speed from the computer's 16-bit processor.

Use of the TI 99/4A as a business aid is, at least for the time being, somewhat limited. Most, but by no means all, business applications require the speed capabilities of a floppy disk storage unit and either a dot matrix or daisywheel printer; all of which may be accessed via the Peripheral Expansion System. When all of these additions are attached, the TI 99/4A is capable of most business applications. If you want to know more about some of the technical words used anywhere in this book, look them up in the glossary at the end of the book.

For any activity where large amounts of information have to be stored, and certain items have to be retrieved (for example, examining the stock-levels of individual items), it is almost essential to use a disk rather than a cassette unit for storage. This is not only because a disk has a greater storage capacity, but also because it permits an item of information to be got out of storage or 'accessed' much more rapidly. Any item stored on a disk can be accessed almost immediately regardless of its position. This is because the 'read head' on a disk unit can move over the surface of the disk to the required point, and for this reason, a disk unit is sometimes called a 'random access' or 'direct access' device. By contrast, using a cassette tape, it is necessary to wind the tape sequentially until the required point is reached. This, of course, can mean long and frustrating delays, and is most irritating when repeated access to information is required.

However, it is nevertheless possible to use the TI 99/4A, even without a disk unit, in a business environment in certain cases. These are primarily where the whole application is loaded into the computer once only every time it is used, and that no further access to the tape is required. A typical example of this type of application is the dynamic spreadsheet analysis program of the type that was first made famous by VisiCalc (see Figure 1.12). Here, mathematical formulae and relationships can be defined directly on the screen and the computer will work out the results. The program is called dynamic because you can change parts immediately. At the time of writing this book, we know of no such program on the market for the TI 99/4A, but there is little doubt that some enterprising programmer will very soon produce one! In fact, the spreadsheet type of program is an ideal application for a command module that simply plugs into the computer, and we may confidently expect Texas Instruments to provide such a program in the near future.

Most computer manufacturers avoid announcing their intentions before the product is available, but at the time of writing this book we noticed sufficient hesitations from the TI spokesman we questioned to have strong suspicions that the company is about to launch a wide range of



```
A5                                                    C!
                                                      34
          A            B            C            D
 1 JAN          FEB          MAR          APR
 2
 3        12.45        14.57        19.56        21.89
 4        33.35        35.67        41.57         51.5
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
```

Figure 1.12 Display produced by VisiCalc.

new software. We suspect, therefore, that by the time you read this book there may be, or may imminently be, a new range of business software including a spreadsheet package and a word processor!

Apart from these last few examples, the majority of such activities can be performed using existing programs. However, if you learn to program the TI 99/4A microcomputer, you can write your own programs. (It is worth remembering that a number of very successful commercial programs were first written because ordinary people at home, at school or in the office, were dissatisfied with the programs they could buy – so they wrote their own). Not only can you express your own ideas, but you can adapt programs that you buy to suit your own requirements more exactly. Since the TI 99/4A computer is much faster at numerical calculations than mere people, it would seem sensible to get the computer to do all your complicated calculations. Besides being used as a source of entertainment, and of educational and business convenience, the TI 99/4A computer can also be used, in this new Information Age, to extend and amplify the human brain.

Summary

The TI 99/4A computer is a small microcomputer which currently comes as a single 16K version (expandable to 48K). The word 'micro' is used to describe the physical size of the machine, and particularly the processor inside, and NOT its ability to compute. However, the smallness of a microcomputer is what makes its computing power available for use as a personal tool at home, at school or in the office. This smallness is a direct result of recent technological developments stimulated mainly by the rivalry between the world's major nations.

The TI 99/4A can be used in many ways, but its main areas of application seem to be for personal entertainment and education, with a certain and lesser amount of business applications likely to develop. Computer programs can already be bought to perform many tasks in these areas. This allows the TI 99/4A to be usefully employed as soon as it is acquired without any expertise in programming being necessary. Clearly, as time progresses, more and more programs will become available.

The capabilities of the TI 99/4A can be extended in a variety of ways by acquiring further units or peripherals, such as a printer, which can then be attached to the computer and used in conjunction with it.

Chapter 2

Using the TI 99/4A microcomputer

Switching on

The TI 99/4A must never be attached directly to the mains power supply. To turn it on, it should first be connected to the supplied power transformer unit, and this in turn plugged into the mains in the usual manner. The ON/OFF button is on the front of the TI 99/4A next to a small light that illuminates when the power is on. It is a slide switch: pushing it to the right will turn the computer ON; pushing it to the left will turn the computer OFF.

However, before you can do anything meaningful with the computer, it must also be attached to a display screen so that you can see what you are doing. This will usually be a domestic television set as described in Chapter 1. Most people will use their own home television set. Although it is preferable to use a colour set, in order to gain the full benefit of the colour capabilities of the TI 99/4A, you can also use one of the more modern black and white sets. A portable black and white set has the added advantage of being easily moveable to any location in the building without being excessively expensive (about £60, for example). Because the computer is American and originally made for the American market, a separate PAL modulator is provided for the UK market. Connecting your TI 99/4A to the display monitor requires only two steps:

1. Connect the single 6-pin plug (called a 'DIN' plug) to the computer console at the video port located on the right-hand side of the rear panel of the TI 99/4A.
2. Disconnect any existing aerial input to the television set, and connect the PAL modulator to the set using the short coaxial cable supplied with the system.

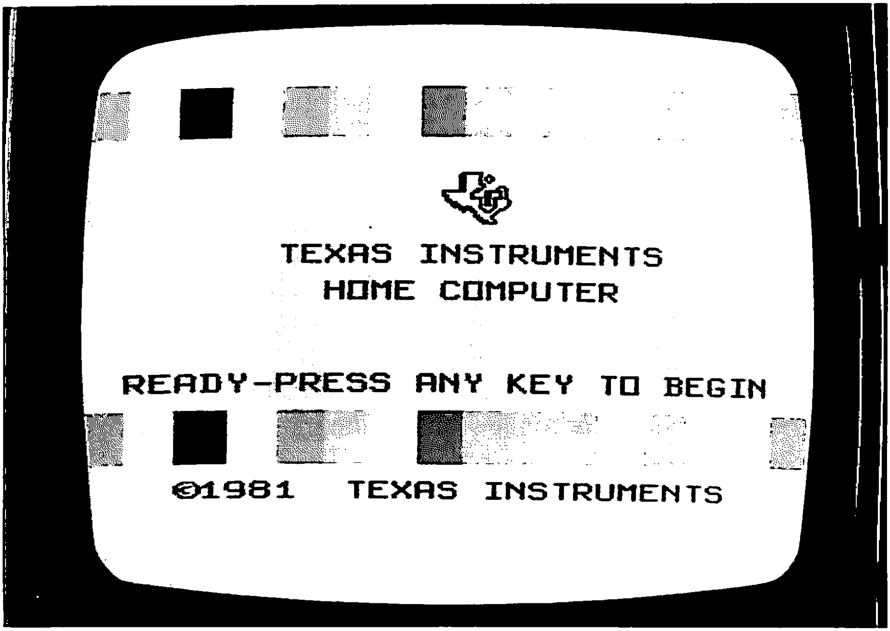
Once the connections have been correctly made you can switch on the computer with the ON/OFF button. To begin with you may get a weak picture of the channel last viewed on the set, that is, BBC 1 or ITV. You must now tune the set to the frequency used by the TI 99/4A. A picture will form when you tune around frequency 36. It will show a blue background (on a colour set) or a light grey background (black and white set). Two horizontal strips of the various colours available cross the screen (showing as varying shades of grey on a black and white set). Between these two strips is the Texas Instruments logo (an outline of the state of Texas inset by the letters 't' and 'i'), and the message:

TEXAS INSTRUMENTS
HOME COMPUTER

This initial display from the TI 99/4A is shown in Figure 2.1. Underneath the bottom strip is a copyright notice from Texas Instruments, while above it is the instruction:

READY – PRESS ANY KEY TO BEGIN

You will find that many different programs give you this instruction at one stage or another. It is probably a good idea to get into the habit of pressing the space bar whenever so instructed. There are two reasons: the first is that it is the largest key on the keyboard and therefore the easiest to find in a hurry; and the second is that this habit will eliminate the risk of hitting a more destructive key, like RESET. There is little danger of this on the TI 99/4A, but we nevertheless make no apologies for the recommendation: now that you have started using a computer, you will find that during the next few years you will come across, and use, many



```
        TEXAS INSTRUMENTS
          HOME COMPUTER


READY-PRESS ANY KEY TO BEGIN

  ©1981   TEXAS INSTRUMENTS
```

Figure 2.1 Screen display when the TI 99/4A is switched on.

others. If you now press the space bar, the screen will clear and you will see a new message. The first two lines consist of the TI logo and the 'Texas Instruments Home Computer' message. Below this is the beginnings of a menu. A computer 'menu' is a list of options that are available together with an individual code (usually a number) that stands for each option. It is rather like the menu in a restaurant. At the moment there is only one option: 'PRESS 1 for TI BASIC'. If, however, you are also using an Extended BASIC command module, then the menu would continue with '2 FOR EXTENDED BASIC'. Now press 1. The screen will clear and be replaced by a system prompt.

The first line tells you that the resident 14K BASIC Interpreter is ready to accept your input. We will learn more about BASIC in Chapter 3. The second line consists of two symbols: a right pointing or closing angle bracket (which some of you might already know better as the 'greater than' symbol), and a flashing block. The first is known as the BASIC Interpreter's 'prompt'. It is always located against the left hand margin of the screen and tells you more specifically that BASIC is ready to receive a new command. The flashing block, known as the cursor, shows you where the next character you type will appear on the screen.

The screen

The letters, symbols and numbers that you can type from the keyboard are all called 'characters'. The screen display is wide enough to take 32 characters on a single line. Once the line is full up with 32 characters, the cursor automatically moves to the beginning of the next line.

The screen display area can hold 24 lines, each of 32 characters. A character can therefore be placed in any of 24x32 (=768) positions on the screen. When the bottom line of the screen is full, the screen contents automatically shift up by one line giving a new blank bottom line. This is called 'scrolling upwards'. The previous top line disappears off the top of the screen.

On some screens, only 28 columns are actually displayed.

The keyboard

The keyboard is very much like any ordinary typewriter keyboard. It is known as a QWERTY keyboard because of the organisation of the top row of letter keys, reading from left to right.

The keys consist of letters, numbers and symbols. There is also a space bar which produces the spaces between words. Using the keyboard is just like using an ordinary typewriter. Pressing any key causes its lower case character to display on the screen, while holding down the <SHIFT>

while pressing any other key will cause the upper case character to appear. However, a word of warning: the upper case characters on this computer are, quite logically, capitals. But so are the lower case characters, albeit small capitals! Sooner or later you will probably hear that BASIC will only recognise capital letters, which is generally true. This is probably why the TI 99/4A returns capital letters in both upper and lower case, because with this computer you can program in both cases. BUT some functions (for example, loading a recorded program from a cassette) insist on the upper case only. Once again, therefore, we recommend that you get straight into the habit of using the ALPHA LOCK key whenever you intend to program in TI BASIC.

There is also a number of other special features which this section will explain.

First of all, note that TI BASIC is designed with an automatic repeat function. This means that if you hold down any key on the keyboard for longer than one second, then that key will repeat automatically. If you have been reading this book with a computer switched on beside you, you may also have noticed by now that in certain circumstances the screen may go blank. Don't worry. You haven't broken the computer nor lost any information it contained. It is simply a device built into the system that will cause the screen to clear if nothing new is entered for a particular length of time. If this has happened to you, press any key to bring back the display as it previously was.

The most important key to note is the <ENTER> key, since this is the most frequently used non-alphanumeric or standard punctuation key. In many ways it is similar to the CARRIAGE RETURN key on an ordinary typewriter since the cursor will move down one line and return to the left hand margin. Its main function on the TI 99/4A, however, is to tell the computer to accept the information you have just finished typing; that is, to 'enter' the information into memory (and, of course, act on it if it is a command).

We have already noted the <ALPHA LOCK>key in passing. It is located in the bottom left hand corner of the keyboard. It is a 'toggle' key. Pressing it once locks all the alphabetical keys into their uppercase mode. The number and function keys are unaffected. When you press the key again, the keyboard returns to normal operation.

There remain two further important keys; keys whose importance lies not so much in themselves but in the effect they have on certain other keys. They are the <CTRL> (control) key to the left of the <SPACE BAR>, and the <FCTN> (function) key to its right. The former is marked with a red dot on the front face of the key, while the latter has a similar white dot. At this point notice a two-level strip overlay above the top row of keys. It refers to this top row of keys. The upper level of the overlay is identified by a red dot and indicates control keys. To activate them simply hold down the <CTRL> key with one hand and press the required control key with the other. The second level of functions is identified by a light grey, almost white, dot. These are function keys that can be activated by holding down the <FCTN> key in a similar fashion. You will notice that a number of the standard character keys have white symbols on their front face: these are also function keys activated by simultaneously pressing the <FCTN> key.

The control characters are used primarily for telecommunications, and will not be discussed in this book. The main function keys, however, are as follows:

F 1 (DEL) This key is used to delete a letter, number or other character from the lines you type.

F = (QUIT) This key returns the system to the master title screen (see Figure 2.1). All the information and/or program that you have entered into the computer will be lost.

F ← (LEFT) Pressing the left-arrow key moves the cursor to the left by one space. The cursor does not erase the characters on the screen as it passes over them.

F → (RIGHT) Pressing the right-arrow key moves the cursor to the right by one space. The cursor does not alter the characters in any way as it passes over them.

F ↑ (UP)
F ↓ (DOWN) } These two keys have various functions depending upon the application for which they are used. You will probably use them mostly when editing a program.

F 2 (INS) This key inserts a letter, number or other character into the lines that you type.

F 3 (ERASE) If you press this key before you press the <ENTER> key, you will erase the line you are currently typing.

F 4 (CLEAR) This key is normally used to clear the screen of any information that you have typed before pressing the <ENTER> key.

Note that the <CLEAR> key is also used to stop a program that is currently being executed. Stopping a program may sound a strange thing to want to do, but in practice you will find that you need to use it quite frequently. It is particularly useful when you start writing and testing your own programs. Using the <CLEAR> key in this manner will not harm the program that is in the computer's memory. It will cause a display something like:

```
  * BREAKPOINT AT 30
>
```

where the number is the program's line number that was being executed at the moment you pressed the <CLEAR> key.

Loading a program

Probably the most enjoyable and painless way to become familiar with the TI 99/4A and its keyboard is to use them to play a game that requires responses from the keyboard. A growing number of games are already available for this.

As we have already seen, there are three common methods of receiving a program (other than typing it in at the keyboard yourself!) for the TI 99/4A. These are via a floppy disk, on a solid state command module, or via a domestic cassette tape. If you have a disk drive with your computer, you are probably already an advanced computer user and there is little that this book for beginners can tell you. We shall not, therefore, discuss floppy disks here.

TI's Solid State Software is probably the most common method of buying ready-to-run software. This is very easy to use. There is a special port on the front right of the console. It has a sprung lid that will automatically close when no module is inserted. Simply plug the module into this port and it is ready to run. There is no need to load the program into the computer's RAM because the module contains its own memory that is used by the computer.

To load any program – irrespective of its name, from a cassette unit to the TI 99/4A, connect the player/recorder to the computer and insert the cassette. At the cassette unit end there are two cables; one with two pins and one with three pins to the connecting lead. If you are only using one lead (and we will assume that you are, since the use of two recorders is really only valid in advanced programming) then it does not matter which of the two cable ends you use (provided you don't try and mix them!). The cable that you don't use will become inactive.

Let us assume that you decide to use the three pin lead. Insert the white pin into the earphone socket (usually labelled 'ear'), and the red pin of the same size into the microphone socket (usually labelled 'mic'). The third and smaller black pin will fit into the socket labelled 'remote', but note that some cassette player/recorders do not have this socket. We are going to suggest that you ignore this third pin altogether at this stage. Let us pretend that you have a cassette with a recording of one of the programs that is used in this book. It is a game known as the Tower of Hanoi, which is called simply 'Hanoi' on the cassette. To load this program into the computer, first make sure all the connecting leads are correctly attached. Rewind the cassette to the beginning (check that you haven't connected the 'remote' pin by mistake), and then type:

```
OLD CS1 <ENTER>
```

Remember that you can only enter a command when the command prompt (>) shows against the left hand margin. The OLD command refers to an old (as opposed to new) program; that is, one that has already been entered and saved. CS1 directs the computer's attention to the cassette port, and <ENTER>, as we have already seen, instructs the computer to obey the line.

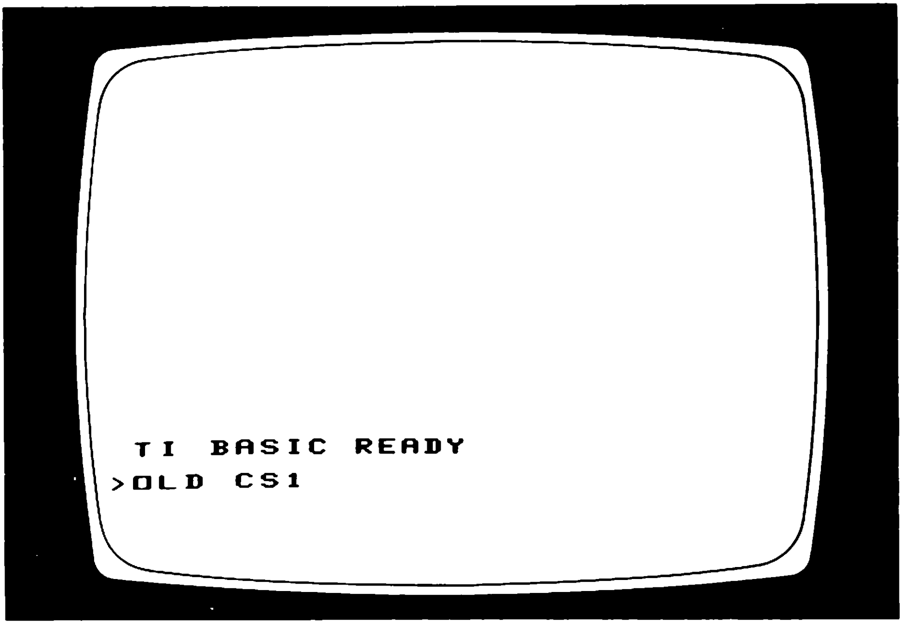
As soon as you press <ENTER>, the screen will display the following message:

```
>OLD CS1
  ★ REWIND CASSETTE TAPE    CS1
    THEN PRESS ENTER
```

The TI 99/4A will now guide you through the process of loading from the cassette. Note that it assumes that you have only the program on the cassette. This is advisable, particularly when you are developing your own programs, but it is not essential. What it means, however, is that you cannot command the computer to find and load a particular program (one of many) recorded on the cassette; it will simply load the next program it finds.

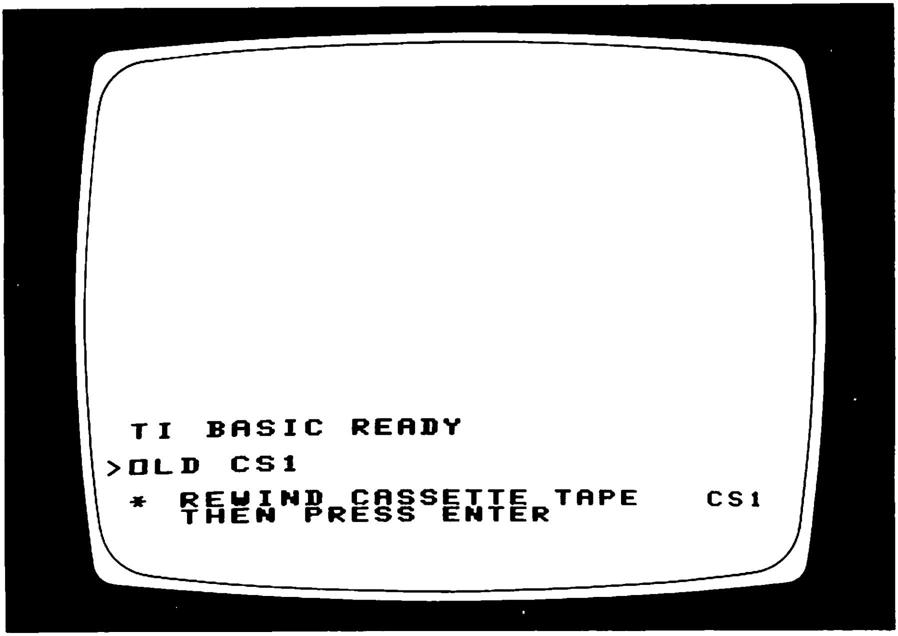
Once the sequence for loading an old program from cassette into the computer's RAM has been successfully completed, the command prompt will again appear at the left margin. You can now run the program merely by typing RUN, and entering this command by pressing the <ENTER> key. If you have loaded a games program, the program itself should now give you instructions on how to play the game. These instructions, or any set of messages between the computer and user, are called a 'dialogue'. The complete dialogue for successfully loading a program is shown in Figures 2.2, 2.3, 2.4, 2.5 and 2.6.

If you use long cassettes with multiple programs (never use C60s or longer since the tape itself tends to be thinner than some of the shorter ones) you may find that you spend a long time trying to find the precise program you wish to load. To avoid these long waits you may consider using the shorter C12 cassettes and having only one or two programs on each cassette. Alternatively you can keep a written record of the general location of the beginning of each program (if your cassette unit has a counter, you can keep an exact record), and finally, you can also detach all the connections from the computer and use the cassette as a simple voice recording unit to speak the name of each program just before it starts on the tape. This way you can use the fast forward and back facilities of the



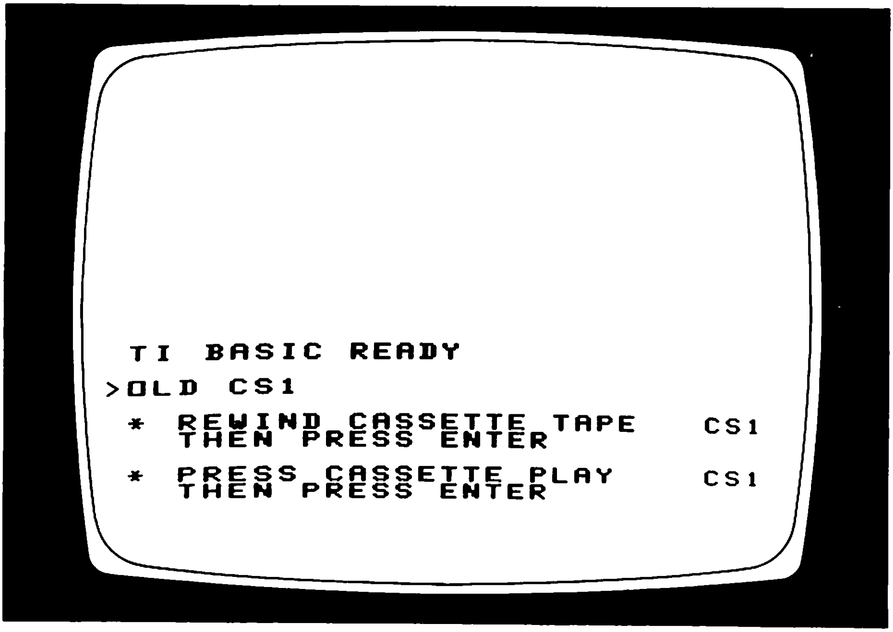
```
TI BASIC READY
>OLD CS1
```

Figure 2.2 Dialogue while loading a cassette from tape: 1.



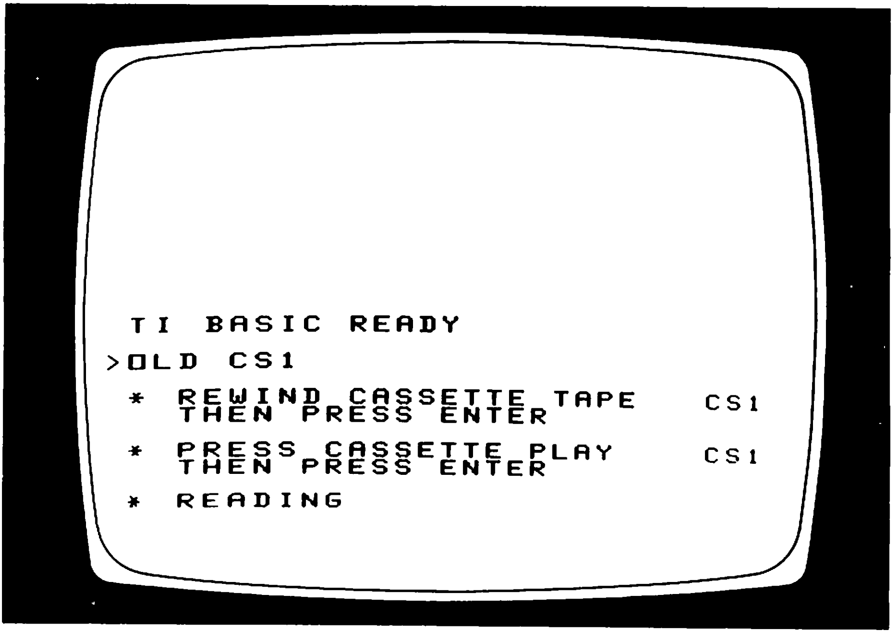
```
TI BASIC READY
>OLD CS1
* REWIND CASSETTE TAPE    CS1
  THEN PRESS ENTER
```

Figure 2.3 Dialogue while loading a cassette from tape: 2.



```
 TI BASIC READY
>OLD CS1
 * REWIND CASSETTE TAPE       CS1
   THEN PRESS ENTER
 * PRESS CASSETTE PLAY        CS1
   THEN PRESS ENTER
```

Figure 2.4 Dialogue while loading a cassette from tape: 3.



```
 TI BASIC READY
>OLD CS1
 * REWIND CASSETTE TAPE       CS1
   THEN PRESS ENTER
 * PRESS CASSETTE PLAY        CS1
   THEN PRESS ENTER
 * READING
```

Figure 2.5 Dialogue while loading a cassette from tape: 4.

cassette unit to listen for the name of the program you want. When you have found the program you are looking for, reconnect the unit to the TI 99/4A, enter:

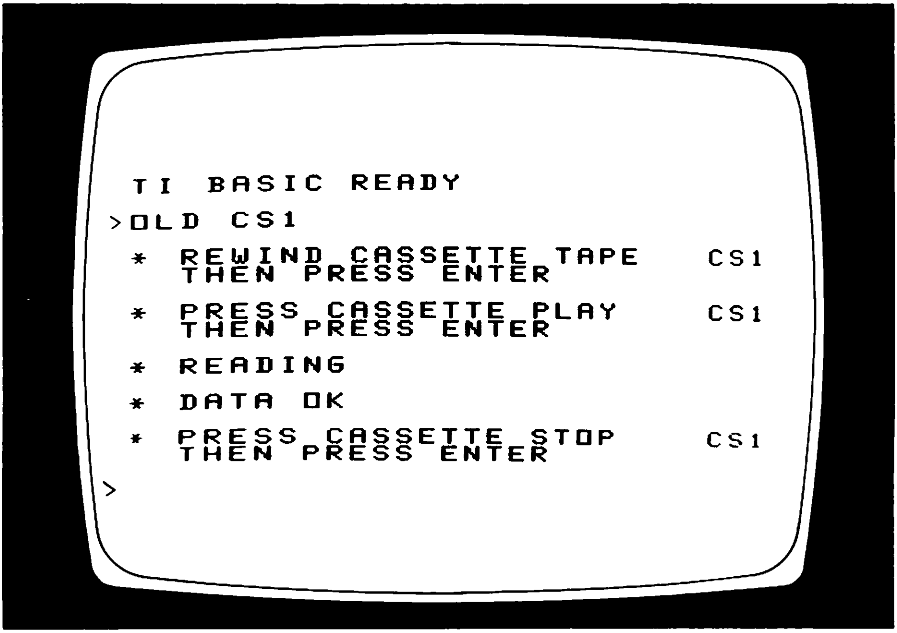
OLD CS1 <ENTER>

at the keyboard, and press the play switch on the cassette unit. Ignore the command to rewind the cassette to the beginning and simply press <ENTER> again. The TI 99/4A will now load the very next program it finds on the cassette.

As a general rule, it is useful to develop individual programs, particularly if they start to get quite long and elaborate, on individual tapes. When you have completely finished the program, you may then store it with several others on a single cassette.

Editing

'Editing' is the name given to correcting or altering a piece of typing. The TI 99/4A has excellent facilities. If you realise that you have made a typing mistake immediately you have made it, the simplest way to put it right is by using the <LEFT ARROW> key. Pressing it once makes the cursor go back a space. It does not yet delete that space, but it does allow you to type



```
 TI BASIC READY
>OLD CS1
 * REWIND CASSETTE TAPE      CS1
   THEN PRESS ENTER
 * PRESS CASSETTE PLAY       CS1
   THEN PRESS ENTER
 * READING
 * DATA OK
 * PRESS CASSETTE STOP       CS1
   THEN PRESS ENTER
>
```

Figure 2.6 Dialogue while loading a cassette from tape: 5.

in the correct characters over the top of the errors.

If you have already entered the line of typing with the mistake in it, you would normally have to re-enter the whole line. Let us say that you are typing a program with the following line:

```
110 PRINT "WHAT APPEARS TO BE THE MATTHER?"
```

Once you have typed <ENTER> at the end of the line, you will be unable to use the <LEFT ARROW> key to correct the word 'MATTHER'. Nevertheless, you still need to remove the incorrect 'H'. You may decide to simply retype the whole line. To do this, re-enter the line correctly, making sure that you start the line with the correct line number. Use this method with care. Every time you start a line of text with a number, the BASIC interpreter inside the TI 99/4A will assume that it is a program line; and that if another with the same number already exists, the new line is to replace it completely. Thus if you enter:

```
100 PRINT 'WHAT APPEARS TO BE THE MATTER?' <ENTER>
```

you will not have changed the incorrect line at 110, but will have changed whatever used to be line 100. Line 110 will remain exactly as it is.

Similarly, if you enter a line number without any text you will delete any existing text on that line (what you will be doing is instructing the TI 99/4A to replace that line with nothing – which is exactly what it will do!). Thus, if you enter '100', then realise that you really want to change line 110, and immediately type '<ENTER>' so that you can start typing line 110, what you are doing is deleting line 100.

It is a better idea to get used to using the TI 99/4A's Editor. An Editor is a small program or routine that will specifically enable you to edit text on the screen. (It is the very simplest and very earliest form of word processor.) To use the Editor, simply type 'EDIT' and the line number, thus:

```
EDIT 110 <ENTER>
```

The screen will print out the whole of the line as follows:

```
EDIT 110
110 PRINT "WHAT APPEARS TO BE THE MATTHER?"
```

Notice that we have underscored the initial 'P' of PRINT. This is to indicate the position of the flashing cursor. Now use the <RIGHT ARROW> key to move the cursor to the required position over the incorrect 'H'. Every time you press this key, the cursor will move one position to the right. As it does so, the characters it passes remain shown on the screen. Thus, if you press the <RIGHT ARROW> key 15 times, the screen will show:

```
EDIT 110
110 PRINT "WHAT APPEARS TO BE THE MATTHER?"
```

Continue pressing the <RIGHT ARROW> key until the cursor is directly over the the 'H' of 'MATTHER'. If you go too far, press the <LEFT ARROW> key to take the cursor backwards to the correct position. Now press the <DEL> (for delete) key. The 'H' will disappear and the text will close up to fill the space.

Let us say, however, that you want to change the line:

```
110 PRINT "WHAT APPEARS TO BE THE MATTHER?"
```

to

```
110 PRINT "WHATEVER APPEARS TO BE THE MATTER?"
```

This time we need to insert characters as well as delete the 'H' in 'MATTHER'. Proceed as above until the screen shows:

```
EDIT 110
110 PRINT "WHAT APPEARS TO BE THE MATTHER?"
```

Now press the <RIGHT ARROW> key until the screen shows:

```
EDIT 110
110 PRINT "WHAT_APPEARS TO BE THE MATTHER?"
```

Type <INS>. This is the Editor command to enter INSERT mode. A computer mode is a condition under which the computer will perform certain functions. Thus, INSERT mode allows the computer to insert text without overwriting existing text. (Another mode that we will look at later in this chapter is IMMEDIATE mode, which allows you to use the TI 99/4A as a calculator.)

Now type 'EVER' so that the screen shows

```
EDIT 110
110 PRINT "WHATEVER_APPEARS TO BE THE MATTHER?"
```

At this point you will need to leave the INSERT mode so that you can use the <RIGHT ARROW> key to move the cursor to the incorrect 'H'. If you do not exit from the INSERT mode, you will never be able to move the cursor to the required position. Every time you press the <SPACE BAR>, for example, you will be inserting a space into the text, and not moving any closer to your goal! Luckily, however, pressing either the <LEFT ARROW> or the <RIGHT ARROW> key will cause the computer to exit from INSERT mode. Now use the <RIGHT ARROW>

key to move the cursor to the 'H'. Delete this with the <DEL> key and then press <ENTER> to tell the TI 99/4A to accept the changes you have made. The screen will then show:

```
EDIT 110
110 PRINT "WHATEVER APPEARS TO BE THE MATTER?"
>_
```

There are a number of other more complicated editing commands that can be used. Details of these can be found in the TI 99/4A's training manual. Briefly, they include the <UP ARROW> key (allows you to edit the line with the next lower number); the <DOWN ARROW> key (allows you to edit the line with the next higher number); and the <CLEAR> key (allows you to ignore any changes you have made so far, and exit from Edit mode). It is well worth remembering this last key, for it will probably save a lot of wasted time and effort when you start writing your own programs!

The TI 99/4A also has two other features that can be of great benefit during editing sessions. These are the NUMBER (NUM) and RESEQUENCE (RES) commands. If you are entering a new program or adding a new subroutine to an existing program, and want to avoid the frustrating problem of forgetting to enter the new line number each time, enter the following command:

```
NUM 500,10 <ENTER>
```

This tells the computer to automatically produce new line numbers every time you press the <ENTER> key. The first number is to be 500, and the numbers are to be incremented by 10 each time. If you just type:

```
NUM <ENTER>
```

then the computer will start to number from line 100, in increments of 10.

The RES command will resequence the line numbers. This is of particular use when you are amending a program and need to add, say, five new lines between lines 100 and 102. The command:

```
RES <ENTER>
```

will resequence all the program's line numbers in increments of 10, starting from the line number 100. You can specify the starting point and the increment value in the same manner as for the NUM command.

Giving simple instructions to the TI 99/4A

Apart from the INSERT mode that we have just talked about, there are two other modes that you must understand. These are IMMEDIATE mode (sometimes known as COMMAND mode, because it is here that you enter a command) and DEFERRED mode. If you enter text into the TI 99/4A in a line starting with a number, the computer will store that line without acting on it. That is, if you start a line with a number, you are automatically using DEFERRED mode. In this case, the computer assumes that you have typed a line of a program that you will want to RUN later on in conjunction with many other lines.

If you do not use a number at the beginning of the line, the computer will automatically enter IMMEDIATE mode. In this case, the TI 99/4A will act on the instructions it receives immediately you press the <ENTER> key. In the rest of this chapter we will look at using the computer in IMMEDIATE mode. In Chapter 3, we will begin to look at using DEFERRED mode: that is, at programming.

Generally speaking, what a computer does in most situations is to accept and store information of some kind, manipulate or process it, and then give the results in some appropriate form. This can be illustrated by instructing the computer to do something with a set of characters. Such a set of characters is called a 'string'. In the BASIC language the dollar sign $ is used to tell the computer that you want it to treat something as a string. For example, the string or characters 'NICHOLAS AND JONATHAN' can be stored in the TI 99/4A's memory by typing and entering the following:

```
A$ = "NICHOLAS AND JONATHAN"
```

You will, of course, remember that you have to press the <ENTER> key to instruct the computer to obey the commands you give it. Pressing <ENTER> causes the TI 99/4A to execute this instruction, which it does by giving the name A$ to a part of its memory in which it stores the string of characters between the inverted commas. This is illustrated in Figure 2.7. This BASIC language instruction is equivalent to the plain English instruction: 'store the string of characters between the inverted commas in a part of the memory, and give it the name A$'. Note that spaces count as characters just as letters do.

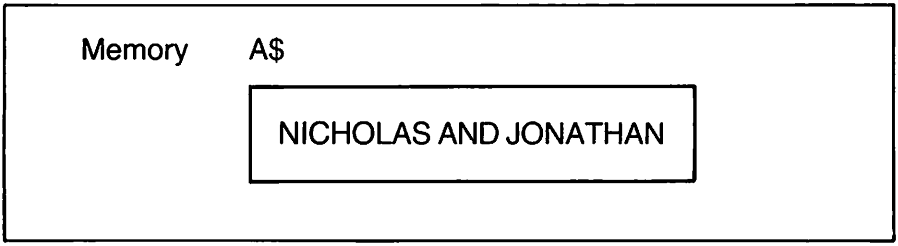


Figure 2.7 A string stored in memory.

When you press <ENTER> and the command is executed, there is no visible outward sign that anything has happened, other than the appearance of '>' prompt at the beginning of the next line. This means that the TI 99/4A is ready to accept your next command. However, there is now an area of the computer memory that is storing the phrase 'NICHOLAS AND JONATHAN'. To demonstrate that the instruction has been correctly obeyed, you now need to know how to get at the information that has just been stored. You can print out the information stored in the part of the memory given the name A$ by typing and entering:

```
PRINT(SPC)A$ <ENTER>
```

(Wherever we use the symbol '(SPC)', it is to indicate that a physical space character **must** be entered at this point.)

In response, the computer will then print on the screen:

```
NICHOLAS AND JONATHAN
```

This BASIC instruction can be understood as 'print out what is stored in A$'.

You can instruct the TI 99/4A to find the number of characters stored in the area of memory called A$ with the LEN instruction, where LEN stands for 'length'. For example, suppose you now type:

```
PRINT LEN(A$)
```

When you press <ENTER>, the number '21' will appear on the screen. In this case the 21 characters consist of 19 letters and two spaces. This instruction can be interpreted as 'print out the length of the string stored in A$', or as 'print out the number of characters stored in the string A$'.

BASIC also has some features that enable you to manipulate strings. The SEG$ command allows you to pick off a number of characters from anywhere within the string. For example, typing:

```
PRINT SEG$(A$,1,8) <ENTER>
```

will cause the computer to print the following eight characters from the string:

```
NICHOLAS
```

That is, starting from the 1st character of the string known as A$, the computer will display the next 8 characters.

Similarly, typing:

```
PRINT SEG$(A$,14,8) <ENTER>
```

gives the eight characters 'JONATHAN' from the right of the string.

This last instruction can be interpreted as 'print the eight characters following the 14th character of the string of characters stored in A$'.

You can see, then, that BASIC programming instructions are a sort of shorthand for instructions expressed in English. But because the TI 99/4A cannot think for itself, all of its instructions must be expressed in precisely the correct way. If there is even a slight error in the way that an instruction is written, no computer yet available will be able to recognise it. The TI 99/4A will, however, let you know that it doesn't understand what you are trying to do by printing an error message on the screen. Here are some examples, with their meanings:

★ INCORRECT STATEMENT This is a syntax error, the most frequent of all errors. Usually caused by a typing or spelling error, or simply not knowing the correct format of the BASIC command.

★ CAN'T CONTINUE This occurs if you try to continue a program that has ended, or does not exist.

★ I/O ERROR This error message is generated when input/output functions fail. A two digit code is also generated. Try to load a program from cassette using the OLD command, but with the keyboard in lower case, or small capitals, mode.

A full list of the error codes can be found in the User's Reference Guide supplied with the TI 99/4A.

We have seen how BASIC makes it easy to dissect strings. In the same way, it is also easy to build them up. To illustrate this you will need to store at least two character strings. Do this first by typing:

```
S$ = "SKY" <ENTER>
T$ = "TRAIN" <ENTER>
```

Now see what you can build up from these strings. Try typing:

```
PRINT S$ & T$ <ENTER>
```

This instruction means 'print the string stored in S$ followed by the string stored in T$'. It produces:

```
SKYTRAIN
```

The following is a more complicated problem, but try to work out for yourself exactly what it tells the computer to do:

```
PRINT SEG$(S$,1,2) & SEG$(T$,4,2) <ENTER>
```

The result is 'SKIN'.

Using the same principle, it is possible to get the TI 99/4A to produce a number of other words from the two strings stored as S$ and T$. Try and make the computer produce the following: 'INKY', 'TRAY', 'STRAIN', 'STINKY'.

The TI 99/4A as a calculator

The TI 99/4A can be used as a calculator. If it seems like a rather expensive calculator, remember that this is only one, and perhaps the least useful, of ways in which it can be used.

As you will by now have realised, the number keys are situated on the top row of the keyboard. There are also five arithmetic keys for you to remember. The four main ones are:

+ means 'add to'
– means 'subtract' or 'take away'
★ means 'multiplied by'
/ means 'divided by'

The fifth instruction is the more advanced arithmetic function of exponentiation. The symbol to use is '↑'. This function is used to raise a number to the power of a second number. Thus, 'squaring' 6 is the same as raising it to the power of 2, which is of course 6 ★ 6, or 36. The way to enter this is:

```
6↑2
```

If you wish to 'cube' the figure, that is, raise it to the power of 3 (6★6★6), use:

```
6↑3
```

For the time being, however, we shall concentrate on the four major functions. The symbol ★ is used for multiplication to avoid confusion with the letter X. The symbol / is used because there is no ÷ on the keyboard, and the alternative fraction display must be expressed on a single line.

Arithmetic calculations can be performed by instructions such as the following:

```
PRINT 2+3+4 <ENTER>
```

The answer 9 is immediately displayed on the next line. A more complicated calculation could be:

```
PRINT (2★3+4)/5 <ENTER>
```

The answer is 2. How you express the calculation instructions is very important, for the computer will always obey a certain sequence when it performs the arithmetic. This sequence is the generally accepted sequence taught in schools and should not, therefore, cause any problems.

The sum is performed basically from left to right; but multiplications and divisions are performed first. Thus:

```
10−2★3
```

will first do 2★3, and then subtract the result from 10. The answer is 4 (and not, as you might otherwise think, 24). You can change this sequence if you wish to by using brackets. Brackets are always calculated first. Thus, if you type:

```
PRINT (10−2) ★ 3 <ENTER>
```

the answer will now be 24.

Numbers can also be stored by using lines such as the following:

```
A = 3 <ENTER>
B = 4 <ENTER>
```

Notice that with numbers, you do not need to use the $ symbol for the location. What you have done, of course, is to store the value of 3 at a location called A, and the value of 4 at a location called B. The result of this operation on the contents of memory is represented in Figure 2.8.

You can now perform calculations on the contents of these locations even if you do not know what the actual values are. For example:

```
PRINT A <ENTER>
```

gives:

```
3
```

Similarly:

```
PRINT A ★ B + 8 <ENTER>
```

gives:

```
20
```

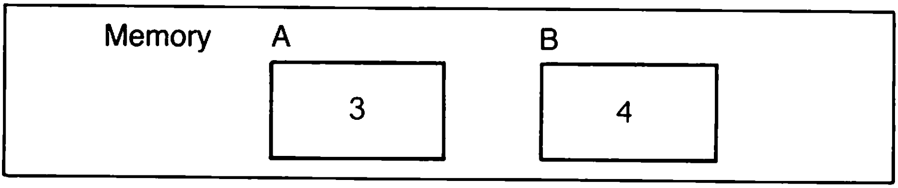


Figure 2.8 Numbers stored in memory after A=3, B=4.

Summary

A good way to become familiar with the TI 99/4A's keyboard is to run a games program which requires responses to be made from the keyboard. A program can be loaded into the computer's memory from a cassette by typing an OLD CS1 command. Once a program is loaded, it can be RUN. So, when starting to use the TI 99/4A computer, it is a good idea to practice with programs already recorded on cassette.

Most of the keys on the keyboard cause the corresponding symbol to appear on the screen when they are pressed, as one would expect. However, a few keys can be used (notably the <FCTN> key) to produce different symbols and or instructions.

Instructions can be given directly to the TI 99/4A in its own TI BASIC language (a version of BASIC produced by Texas Instruments). With the aid of a small repertoire of instructions it is able to make the computer perform such diverse activities as storing and manipulating words, and storing and performing calculations on numbers.

Self-test questions

1. What is the instruction which starts the procedure for loading a program into the TI 99/4A computer from a cassette?
2. How do you start the Editor so that you can edit a program line? Using as few instructions as possible, change the line:

   ```
   100 THE ASCENT OF MAN
   ```

   to

   ```
   100 THE ANCIENT OMEN
   ```
3. Make the computer store the words 'LEAD' and 'POSE' in its memory. Using these two stored words only, write the instructions that will cause the TI 99/4A to display:

   ```
   PLEAD
   POSE
   PLEASE
   LOSE
   ADDLE
   ```
4. Make the TI 99/4A store the numbers 4 and 5 in its memory. Using these stored numbers only, enter the instructions necessary to make the computer produce the results 16, 24 and 36.

Chapter 3

Introduction to programming

Writing and running a simple program

Towards the end of Chapter 2 we looked at using the TI 99/4A in IMMEDIATE mode; that is, we gave it single line instructions that were to be obeyed immediately. In this chapter we shall examine the alternative DEFERRED mode, which is another way of saying: 'programming'. In DEFERRED mode, all instructions must be on a line starting with a number. The computer will then store these commands until it is later told to RUN them. At this point it will obey each stored line of instructions in the order of the numbers at the beginning of the line.

A program, then, is a sequence of commands for the TI 99/4A to obey. The language in which the commands must be written in order that the TI 99/4A can respond to them is BASIC, and a few examples of individual BASIC commands have already been introduced in the previous chapter. BASIC is a simple programming language that was devised at Dartmouth College in the USA, and which first came into use in the early 1960s. It was intended to be easy to learn and easy to teach: and, indeed, its overwhelming popularity as a language for microcomputers stems from the very fact that it is very easy to learn.

The TI 99/4A first deals with a program by storing it, so that it can then execute it when instructed to do so. When a program is stored in the TI 99/4A's memory it can be executed as many times as desired, or it can be modified prior to running it again. When a BASIC command is preceded by a number, the TI 99/4A treats that line as an instruction belonging to a BASIC program and stores both the number and the command. When the line is first entered, the command is not immediately executed, but is stored so that it can be executed later. A TI 99/4A program consists of a set of numbered commands. The numbers give the order in which the commands are to be executed when the program is run. The command or commands on the line with the lowest number are the ones to be executed first, and so on in ascending order. In fact, the instructions that make up a program can be entered in any order because the TI 99/4A uses the line numbers to put the instructions in the correct order.

To summarise this: a program line consists of a number followed by a command or commands; the TI 99/4A stores these instructions, and puts them in the correct order by using their line numbers; a program consists of a set of instructions; when a program is executed, each instruction is dealt with in sequence by executing its command part.

Now let us write a short program to store the three words 'THE', 'DOG', and 'SHOW', and to use these stored words to write out the phrases 'THE DOG SHOW', 'SHOW THE DOG' and 'DOG THE SHOW'. Before starting it is a good idea to type:

```
NEW <ENTER>
```

because this clears any program previously stored in the TI 99/4A. The screen will revert to the initial BASIC display:

```
TI BASIC READY
>■
```

Now type in the following program exactly as shown, pressing the <ENTER> key at the end of each line to cause it to be stored in the computer's memory (note that in all of the example programs included in this book, the symbol '(SPC)' indicates that you must here type a single

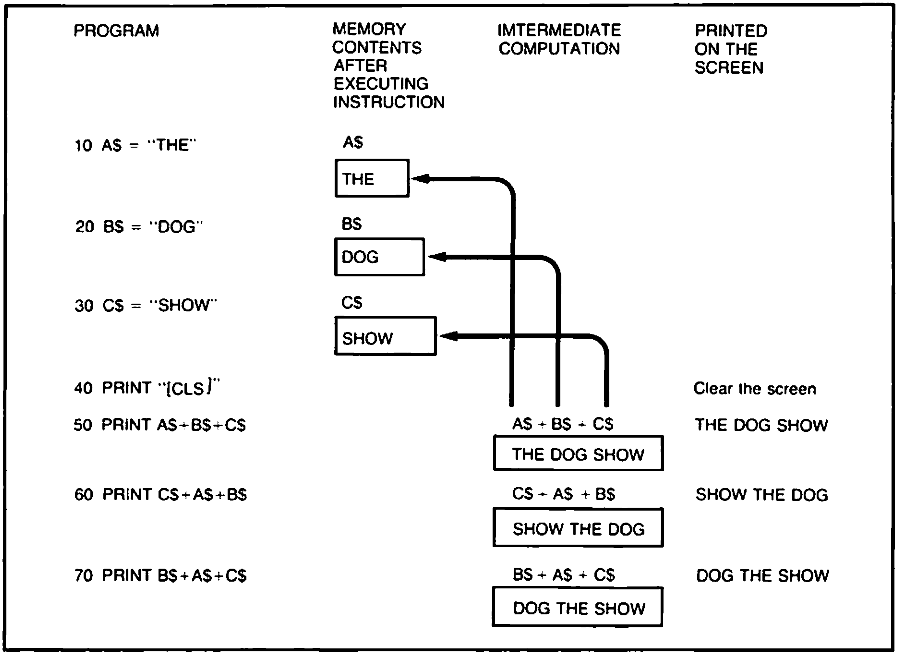


Figure 3.1 The results of running a program.

blank space):

```
10 A$= "THE(SPC)" <ENTER>
20 B$= "DOG(SPC)" <ENTER>
30 C$= "SHOW(SPC)" <ENTER>
40 CALL CLEAR
50 PRINT A$ & B$ & C$ <ENTER>
60 PRINT C$ & A$ & B$ <ENTER>
70 PRINT B$ & A$ & C$ <ENTER>
```

In this program, the three words are stored at lines 10 to 30. Note that a space is included at the end of each word to act as a word separator when the phrases are printed. Line 40 causes the screen to be cleared when executed. It is a good general rule to get into the habit of using this command at the beginning of all of your programs.

Lines 50 to 70 cause the required phrases to be printed. Figure 3.1 shows the consequence of executing each instruction of the program. The result of executing the program is the cumulative effect produced by executing all of its instructions.

To demonstrate that the program has been stored by the TI 99/4A, type:

```
LIST <ENTER>
```

in response to which the TI 99/4A will always produce a 'listing' (a printout) of the program it is currently storing. Check the listing given on the TI 99/4A screen against the listing above to see that they agree exactly and, if they do, execute the program by typing:

```
RUN <ENTER>
```

You will then see the results of the program, looking like this:

```
THE DOG SHOW
SHOW THE DOG
DOG THE SHOW
>■
```

Remember that because the program is stored in the TI 99/4A it can be executed or listed as often as you like. If you wish to list only a part, or just an individual line of the program, type LIST and the line number, or 'LIST from-to'. If a line of the program has been entered incorrectly, it can be corrected by listing it on the screen and using the normal editing procedures as described in Chapter 2. For example, if line 20 has been incorrectly entered, it can be listed by typing:

```
LIST 20 <ENTER>
```

which could give, say,

```
20 B$ = "DIG"
```

Because this is a short line, it could be corrected most quickly by simply retyping the whole line. If it were a longer line, you should use the TI 99/4A's Editor as described in the previous chapter. The corrected program can then be RUN as described above.

To delete a complete line, all that is necessary is to type the number of the line to be removed followed by <ENTER>.

Try typing:

```
60 <ENTER>
```

and then list the program to see the effect it has had. When you have finished experimenting with the program, type:

```
NEW <ENTER>
```

to delete it. After typing this, the LIST command evokes no response. Try it.

Some more BASIC instructions

In this section we meet a few more BASIC instructions. They are incorporated in short programs to illustrate their usefulness. The fact that the <ENTER> key has to be pressed after a command or at the end of an instruction to cause the TI 99/4A to take the appropriate action will not be mentioned explicitly any more. So, as a last reminder, if you have typed something out and nothing appears to be happening as you sit and wait, it may well be that you have not pressed <ENTER>!

Input

Suppose that we should like to modify the program given in the first section of this chapter so that it accepts any three words we might care to give it when we run the program, and then prints out the first followed by the second and then the third; then the third followed by the first and the second; and finally the second followed by the first and the third. The new instruction that we need in order to make the program accept an input is INPUT. When an INPUT instruction is executed it causes a question mark to be printed on the screen to indicate that a response is required, and then stops the program execution and makes the TI 99/4A wait until the user types a response which it can accept. Thus, the instruction:

```
10 INPUT A$
```

produces the question mark on the screen.

If you then type:

```
THE <ENTER>
```

the word THE is accepted and stored in A$. So, in this case, the effect is the same as that of the instruction

```
10 A$ = “THE”
```

The difference is that the latter will assign ‘THE’ to the variable A$, whereas the former can assign whatever you type after the question mark.

An example program can be based on the previous program by replacing the lines that store the three words with three INPUT instructions – one for each word. When the words have been entered in this way, lines 50 to 70 will print out the phrases as before. Note, however, that when a word is entered with an INPUT command, it is not possible to include a space at the end of it. For this reason the spaces to separate the words in a phrase must be placed in the PRINT commands. In this way, the following program for accepting any three words and printing three phrases involving them is obtained:

```
10 CALL CLEAR
20 INPUT A$
30 INPUT B$
40 INPUT C$
50 PRINT A$ & “(SPC)” & B$ & “(SPC)” & C$
60 PRINT C$ & “(SPC)” & A$ & “(SPC)” & B$
70 PRINT B$ & “(SPC)” & A$ & “(SPC)” & C$
```

When this program is executed it could result in a dialogue like this:

```
? THE
? KIT
? BAG
THE KIT BAG
BAG THE KIT
KIT THE BAG
>■
```

Decisions

The TI 99/4A can be programmed to make decisions. This ability can be used to produce some very interesting and powerful programs. The command which permits decision-making uses the BASIC words IF and THEN. It has the form:

```
IF condition THEN command
```

In the condition part of this instruction, both variables and/or values can be compared, typically to see if they are the same or if they differ. The command part must be another BASIC command; for example, an assignment or a PRINT command. When the IF/THEN command is executed, the condition part is first tested. Only if it proves positive will the command part be executed. If the condition part does not hold, then the instruction part is ignored. An example of this type of command is:

```
IF N$ = "PASSWORD" THEN PRINT "ACCEPTED"
```

When this is executed, the TI 99/4A tests to see if the most recent assignment to N$ is PASSWORD: if it is, then ACCEPTED is printed out. If it is not, nothing is done. A second example is

```
IF N$ <> "PASSWORD" THEN PRINT "REJECTED"
```

In this example the pair of symbols <> means ‘not equal to’. (The < symbol on its own means ‘less than’, while the > symbol on its own means ‘greater than’. If something is either less than or greater than the subject, then the only thing it is not, is ‘equal to’ the subject; that is, it is ‘not equal to’.) Thus, when this command is executed, REJECTED is printed only if the most recent assignment to N$ is not PASSWORD.

Now consider a short program to create a sum, display it, accept an answer to it and decide if the answer is correct or not before printing an appropriate message. This requirement is also shown in Figure 3.2, which is an example of a flow chart. The program starts by storing two numbers in A and B, and then line 30 uses these numbers to print out a question about their sum. The question that is printed is WHAT IS 2+3? Using a semicolon to separate the items in a PRINT command gives a spacing different to that produced when a comma is used. Having set a problem, the program accepts an answer at line 40, storing it in C. Then in line 50 the offered answer is tested to see if it is equal to the right answer, and if it is, then an encouraging message is printed. The final line detects when a wrong answer is given and causes the real answer to be printed out on these occasions. The program is:

```
10 A = 2
20 B = 3
30 PRINT "WHAT IS(SPC)";A;"+";B;"?"
40 INPUT C
50 IF C <> A + B THEN 80
60 PRINT "GOOD. THAT IS CORRECT."
70 STOP
80 PRINT "NO. THE ANSWER IS(SPC)";A + B
```

This program can be adapted to give you more than one attempt to find the answer to the way illustrated in Figure 3.3 by using the GOTO command. The command:

```
GOTO 30
```

instructs the program to go to line 30 and execute that line next. A program that expects the user to keep attempting to answer until the correct answer is given is obtained by altering line 50 so that it causes the last line of the program (line 80) to be executed next if the correct answer is given. The last line supplies the reinforcing message of encouragement. If the correct answer is not provided, then line 60 is executed. This indicates that the

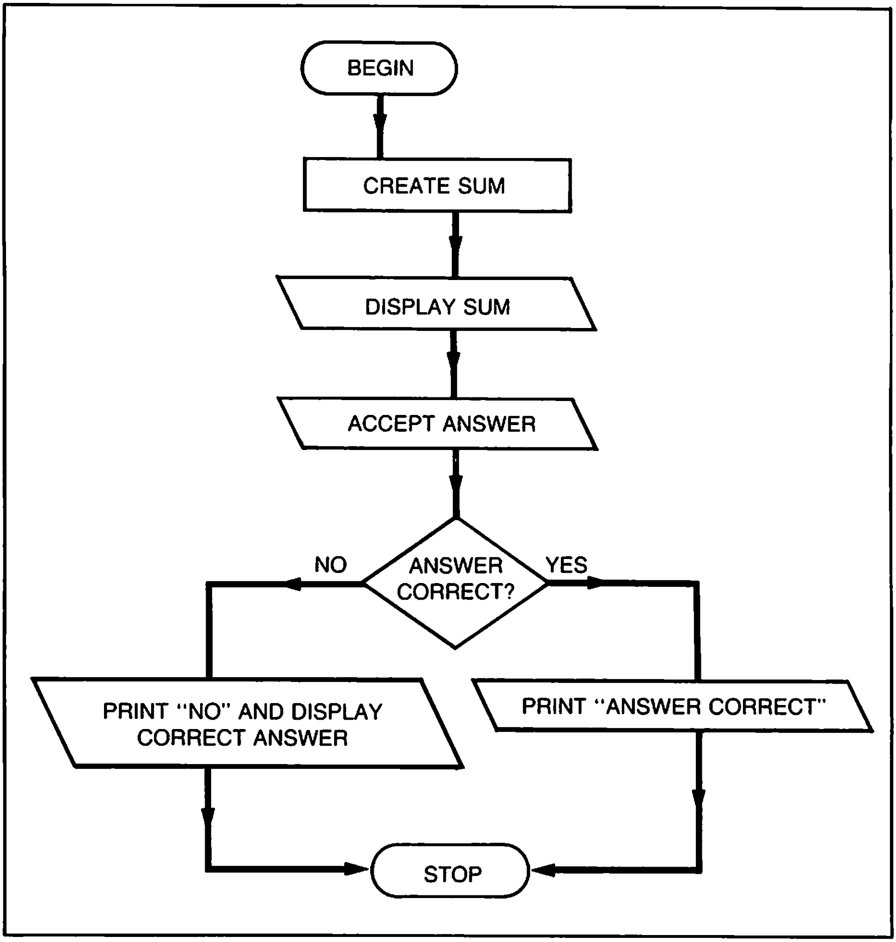


Figure 3.2 Flow chart for simple maths drill.

answer is wrong before line 70 causes a jump back to line 30, so that the question is posed again and a further opportunity to answer is given. This more sophisticated program is listed below:

```
10 A = 2
20 B = 3
30 PRINT “WHAT IS(SPC)”;A;“+”;B;“?”
40 INPUT C
50 IF C = A + B THEN 80
60 PRINT “SORRY. WRONG ANSWER. TRY AGAIN.”
70 GOTO 30
80 PRINT “GOOD, THAT IS CORRECT.”
```

A typical dialogue produced by this program could be:

```
WHAT IS 2 + 3 ?
? 6
SORRY, WRONG ANSWER. TRY AGAIN.
```

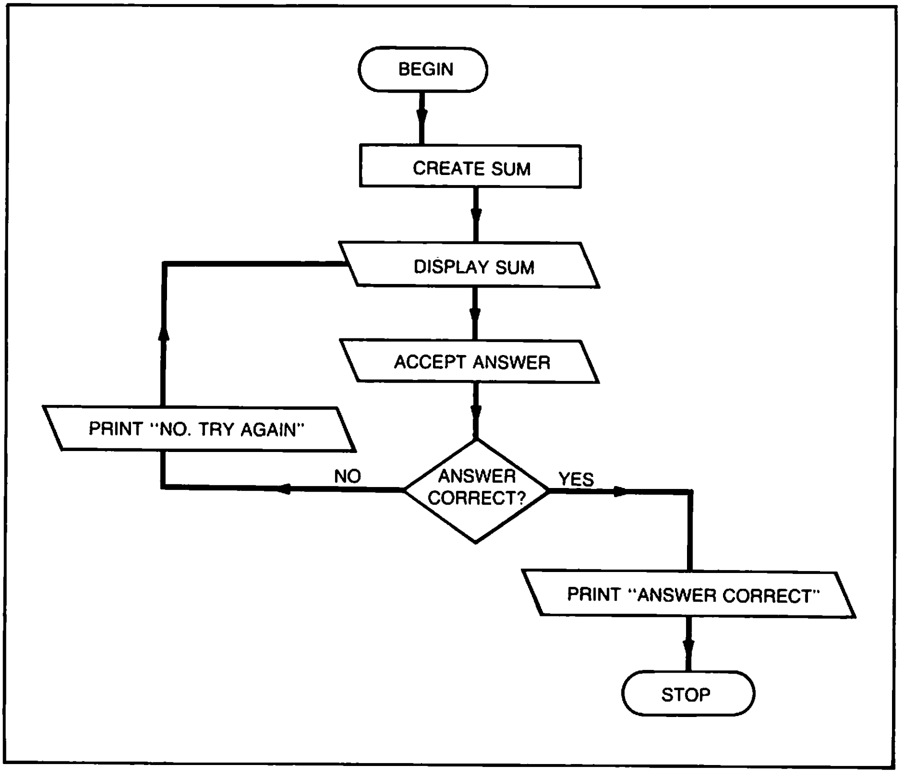


Figure 3.3 Flow chart for improved maths drill.

```
WHAT IS 2 + 3 ?
? 4
SORRY, WRONG ANSWER. TRY AGAIN.
WHAT IS 2 + 3 ?
? 5
GOOD, THAT IS CORRECT.
```

Now try changing line 30 to:

```
30 PRINT “WHAT IS(SPC)”;A;“(SPC)+(SPC)”;B;
```

This time, you have not included a question mark to be printed by the instruction. Notice, however, that the semicolon at the end of the line ‘pulls up’ the question mark produced by the INPUT command. In this way you can eliminate the occurrence of double question marks and make the question and answer sequence more visually acceptable. A typical dialogue might now be:

```
WHAT IS 2 + 3 ? 6
SORRY, WRONG ANSWER. TRY AGAIN.
WHAT IS 2 + 3 ? 4
SORRY, WRONG ANSWER. TRY AGAIN.
WHAT IS 2 + 3 ? 5
GOOD, THAT IS CORRECT.
```

Repetition

The previous program has shown that the TI 99/4A can be programmed to do things repeatedly by using the GOTO command. The mathematical problem is posed repeatedly until the correct answer is given. BASIC, however, has a more direct way to achieve the effect of repetition: the use of the BASIC FOR . . . NEXT command. To illustrate its use, enter the following program:

```
10 FOR I = 1 TO 16
20 PRINT “JOANNE”
30 NEXT I
```

This program will cause ‘JOANNE’ to be printed sixteen times, because all the instructions between FOR and NEXT are repeated as many times as directed by the FOR instruction. In this case, line 10 becomes a counter that goes up by one each time the command ‘PRINT “JOANNE”’ is executed. Notice that the program is going round in circles. This is called a ‘loop’. Line 30 instructs the computer to loop back to line 10 until the counter matches the number specified in the ‘TO’ statement; that is, 16. At this point, the program will come out of the loop and go on to execute the next command, if there is one. The next program illustrates that you can put as many instructions as you want between the FOR and NEXT statements:

```
10 FOR K = 1 TO 9
20 PRINT "REPETITION NUMBER";K
30 PRINT "FRANCES"
40 PRINT
50 NEXT K
```

This causes nine repetitions and produces the output:

```
REPETITION NUMBER 1
FRANCES

REPETITION NUMBER 2
FRANCES
```

and continues up to the ninth repetition.

Now try a program that accepts a word and spells out its letters one at a time. The program must accept the word, find its length and then repeatedly pick out and print the first letter, second letter, and so on, up to the last letter. We have already seen how to separate letters from the left or right of a word, using SEG$, so you now know all the commands necessary to write the program. It will start with a polite request to enter a word, which will be followed by an INPUT command to accept and store the word in W$. At line 30 the number of letters in the entered word is found, and stored in L. Note that line 50 contains SEG$(W$,I,1), and that I has already been assigned the variable loop counter of a FOR . . . NEXT command that is as long as the length of the word: in other words, I is at first letter one of the word, then letter two, etc. Now SEG$(W$,3,1) will find the string of letters in the word stored in W$ which starts with the third letter and is one letter long, which means that it will find the third letter of the stored word. The effect of lines 40 to 60 is therefore to find repeatedly the successive letters of the word and to print out, on the first loop, that 'letter number 1' is whatever the first letter is, and so on. The entire program for spelling out the letters that make up a word is:

```
 5 CALL CLEAR
10 PRINT "ENTER A WORD, PLEASE."
20 INPUT W$
30 L = LEN(W$)
40 FOR I = 1 TO L
50 PRINT "LETTER NUMBER(SPC)";I;
    "(SPC) IS (SPC)"; SEG$(W$,I,1)
60 NEXT I
```

More programs

Let us now write a program to accept any name written in the form:

```
JAMES JOYCE
```

and to produce the output:

```
YOUR FIRST NAME IS JAMES
YOUR SECOND NAME IS JOYCE
```

This may seem at first sight very easy, since after:

```
INPUT N$
```

the first name could be found by SEG$(N$,1,5) and the surname by SEG$(N$,7,5). Unfortunately this will produce nonsense if the name entered is WILLIAM SHAKESPEARE (or any name that is not a combination of two five letter names). The trick is, of course, to locate the position of the space separating the two parts of the name. Then, assuming that the name has been entered correctly, everything to the left of the space is the first name and everything to the right is the surname. If the name is not entered in the way we expect strange results can still be printed, so it is sensible to ask that the name be entered in a standard fashion and then to check it, rejecting it if it does not conform. This reasoning leads us to the following program:

```
 10 CALL CLEAR
 20 PRINT “ENTER YOUR NAME, PLEASE. TYPE”
 30 PRINT “YOUR FIRST NAME, THEN ONE”
 40 PRINT “SPACE THEN YOUR SECOND NAME.”
 50 INPUT N$
 60 L = LEN(N$)
 65 C = 0
 70 FOR I = 1 TO L
 80 IF SEG$(N$,I,1) <> “(SPC)” THEN 100
 90 C =C+1
100 NEXT 1
110 IF C=1 THEN 140
120 PRINT “PLEASE ENTER YOUR NAME AS REQUESTED”
130 GOTO 20
140 FOR J = 1 TO L
150 IF SEG$(N$,J,1) <> “(SPC)” THEN 170
160 B = J
170 NEXT J
180 PRINT “YOUR FIRST NAME IS(SPC)”;SEG$(N$,1,B−1)
190 PRINT “YOUR SURNAME IS(SPC)”;SEG$(N$,B+1,L−B)
```

In this program the screen is first cleared, before lines 20 to 40 display on the screen the instructions for using the program. Line 50 accepts a name and stores it in N$. Line 60 stores the number of characters in the name in L. Lines 70 to 100 scan each character of the name, counting the number of spaces. At the end of the repetitions the number of spaces in the name is held in C, which was set at zero in line 65. If there is not exactly one space in the name, then lines 100 to 130 indicate that the entry is not satisfactory and cause a return to line 20 to permit the name to be entered again. Lines 140 to 170 locate the position of the space, storing it in B, so that line 180 can print all the characters to the left of the space as the first name and line 190 can print all those to the right as the surname.

Our next program produces a rather fascinating mobile display of a worm-like object that moves backwards and forwards across the screen! Although there are other ways in which this effect could be achieved, this method concentrates on the commands that we have already examined. It does, however, introduce a new command: CALL HCHAR. The full format for this instruction is:

```
CALL HCHAR (row number, column number,
                  character code, number of repetitions)
```

Imagine the screen as a large grid made up of 32 (columns) by 24 (lines). The total number of points on this grid is therefore 768, which can be accessed by the TI 99/4A as X,Y co-ordinates with the row being in the range of 1-24 and the column being in the range of 1-32. Thus the top left position is 1,1: and the bottom right is 24,32.

The character codes may be any number between 0 and 32767, but the computer will always convert them to a value between 0 and 255. The character codes between 32 and 127 are defined as standard ASCII codes. See the next chapter on Graphics for more information on CALL HCHAR and the ASCII codes.

Notice also that we have introduced an extension to the FOR . . . NEXT command. At line 60, we have FOR . . . NEXT . . . STEP. If we do not include a STEP value, the computer will assume that we wish to use STEP +1. In other words, the FOR . . . NEXT counter will go up by one in each repetition loop. Here, however, the STEP is −1, and the counter will consequently decrease by 1 for each loop. The mobile display program is:

```
10 CALL CLEAR
20 FOR I = 3 TO 27
30 CALL HCHAR(5,I-1,32,1)
40 CALL HCHAR(5,I,42,3)
50 NEXT I
```

```
 60 FOR I = 27 TO 3 STEP -1
 70 CALL HCHAR(5,I,42,3)
 80 CALL HCHAR(5,I+2,32,1)
 90 NEXT I
100 GOTO 20
```

Because the last line of the program always causes line 20 to be executed again, starting another pass across the screen and back by the worm, the program runs indefinitely when executed. To stop it, it is necessary to press the <CLEAR> key.

The character code 32 in lines 30 and 80 produce a space. This is necessary to prevent the worm leaving a trail across the screen. Try removing line 30 and see what happens.

We hope you will experiment with all the programs throughout this book, and make changes here and there to see what effect they have. On this last program, for example, try a new command: RND. This command will make the TI 99/4A produce a random number between 0 and 1. By multiplying the random number with an upper limit value, and then extracting the integer part, you can create a whole random number up to a maximum value that you can specify yourself. If you wish to make the range inclusive, you need only add 1 to the result. Make the following changes and see what happens. Try to work out for yourself exactly what the program is now doing.

Change all the references to the HCHAR line number in lines 30, 40, 70, and 80 to X. Now add the new line:

```
15 X = INT(RND★23)+1
```

Finally, change line 100 to:

```
100 GOTO 10
```

Run this program and see what happens. Now change line 100 to:

```
100 GOTO 15
```

and see what happens. Try to work out the difference it makes.

We shall now develop a program to translate French words into their English equivalents. To do this, we shall need to store French words and the corresponding English words in such a way that they can be related to one another. BASIC provides the 'array' which is useful for doing this. The single BASIC command:

```
DIM A$(20)
```

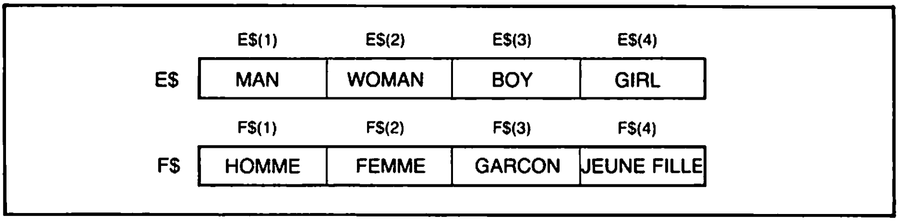
tells the TI 99/4A that 20 variables are to be established as an array, and that their names are to be A$(1) to A$(20). Once established, these array variables can be used in the same way as ordinary variable names. For example, we can make the assignment:

A$(6) = "MAN"

The DIM command also reserves storage space for all the variables in the array. As the following program shows, arrays can be used to great advantage in FOR . . . NEXT repetitions. In this program we shall use two arrays, as illustrated in Figure 3.4, with one (F$) holding French words and the other (E$) holding the equivalent English words in the same order.

The program translates by seeking to match the French word to be translated with one of the French words stored in the array F$. If a match is found then the word is associated with the English word in the corresponding position in the array E$. In the program line 10 reserves space for the two arrays which will hold the French and the English words. The words themselves are stored by lines 20 to 55. When the lines up to 55 have been executed, the state of the memory is as shown in Figure 3.4. Line 60 requests the entry of a French word and line 70 accepts and stores it. Lines 80 to 120 establish a loop that examines each of the words in the 'French' array. If the French word is found, then line 100 prints the corresponding English word on the screen, and line 110 redirects the program back to line 60. If no such French word is found in the array, then line 130 prints a statement to that effect, before once again sending the program back to line 60.

```
10 DIM E$(4), F$(4)
20 E$(1) = "MAN"
25 F$(1) = "HOMME"
30 E$(2) = "WOMAN"
35 F$(2) = "FEMME"
40 E$(3) = "BOY"
45 F$(3) = "GARCON"
50 E$(4) = "GIRL"
55 F$(4) = "JEUNE FILLE"
```



	E$(1)	E$(2)	E$(3)	E$(4)
E$	MAN	WOMAN	BOY	GIRL
	F$(1)	F$(2)	F$(3)	F$(4)
F$	HOMME	FEMME	GARCON	JEUNE FILLE

Figure 3.4 Two parallel arrays for translation program.

```
 60 PRINT "ENTER FRENCH WORD."
 70 INPUT W$
 80 FOR I = 1 TO 4
 90 IF W$<>F$(I) THEN 120
100 PRINT E$(I)
110 GOTO 60
120 NEXT I
130 PRINT W$;"(SPC)IS NOT IN MY VOCABULARY"
140 GOTO 60
```

Clearly, as presented here, this program has a very limited vocabulary. It can, however, be extended in a very simple manner (by adding more lines of the type shown at lines 20 to 55 and making other straightforward adjustments). Also it is not difficult to adapt the program so that it translates from English to French. As we have already mentioned, all of the programs presented in this section are intended to be used as vehicles for experimenting with programming in BASIC. They can be amended, extended and improved in many ways.

Saving programs

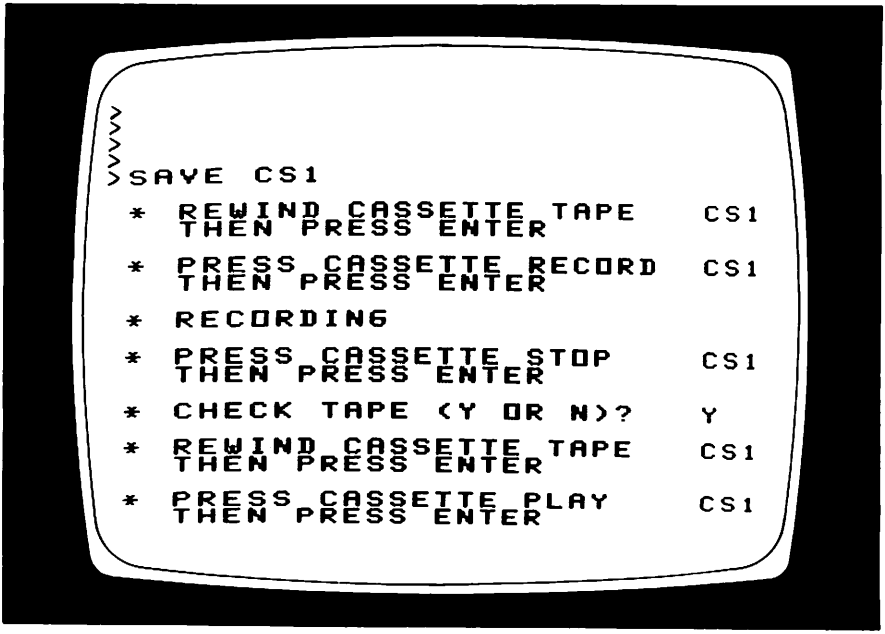
When the TI 99/4A is switched off, the program stored in its internal memory is lost. To avoid having to type in the same program every time you want to use it, it is necessary to copy it on to some form of permanent storage. A program that is stored in the TI 99/4A can be permanently saved using either a cassette or a disk unit (if you have one) so that it can be loaded again later. Since you need the Peripheral Expansion System to be able to use a disk drive, we shall not discuss saving on to disk in this book for beginners. We shall simply concentrate on saving a program on to a cassette tape.

To save the program stored in the TI 99/4A's internal memory on to a cassette tape, first ensure that the cassette unit is correctly attached to the TI 99/4A and then put a tape cassette (preferably blank) into it. Completely rewind the tape and then wind it forward a little to avoid trying to record on the tape leader. When the tape is positioned properly, enter the simple command:

```
SAVE CS1
```

The computer will now begin giving instructions on the screen to help you through the SAVE procedures (see Figure 3.5). If you follow these instructions completely, you should have little difficulty in successfully saving a program on to cassette. Note that when the computer has finished recording, it gives you the option to check the data on the tape with the original program still held in memory. It is a simple procedure and one well worth adopting every time you save on to cassette. The full sequence of commands, including the verification process, is as follows (note that the only response you have to make is shown in heavy type):

```
>SAVE CS1
  ★ REWIND CASSETTE TAPE    CS1
    THEN PRESS ENTER
  ★ PRESS CASSETTE RECORD   CS1
    THEN PRESS ENTER
  ★ RECORDING
  ★ PRESS CASSETTE STOP     CS1
    THEN PRESS ENTER
  ★ CHECK TAPE (Y OR N)?    Y
  ★ REWIND CASSETTE TAPE    CS1
    THEN PRESS ENTER
  ★ PRESS CASSETTE PLAY     CS1
    THEN PRESS ENTER
  ★ CHECKING
  ★ DATA OK
  ★ PRESS CASSETTE STOP     CS1
    THEN PRESS ENTER
```



```
>
>
>
>
>SAVE CS1
 *  REWIND CASSETTE TAPE        CS1
    THEN PRESS ENTER
 *  PRESS CASSETTE RECORD       CS1
    THEN PRESS ENTER
 *  RECORDING
 *  PRESS CASSETTE STOP         CS1
    THEN PRESS ENTER
 *  CHECK TAPE (Y OR N)?        Y
 *  REWIND CASSETTE TAPE        CS1
    THEN PRESS ENTER
 *  PRESS CASSETTE PLAY         CS1
    THEN PRESS ENTER
```

Figure 3.5 Dialogue after saving a program.

We have already seen how to load a program from tape into the computer's memory. On nearly all microcomputers, the level at which the volume is set on the player/recorder is critical in both saving and loading. If the level is incorrectly set (and to begin with, it probably will be!), the screen will probably produce some form of I/O error message (refer to the User's Reference Manual for details of the different messages), when you try to load, and perhaps nothing at all when you try to save. For this reason we recommend that when you first start using the TI 99/4A you purchase a cassette with a program already recorded on it. Keep trying to load the program until you succeed. To begin with, try setting the volume to approximately mid-range. Each time you fail to load successfully, move the volume up slightly and try again. When you find the right level, leave the recorder set at that point.

Summary

The TI 99/4A can store a BASIC program which can then be executed as often as required, or which can be modifed before it is run again. The TI 99/4A's BASIC language is a simple English-like language which provides, among other things, facilities for storing and manipulating information, making decisions and for repeating an action as often as necessary. In this chapter these facilities are introduced and incorporated in simple programs to illustrate ways in which they can be used. When a program has been written it can be saved on cassette or disk, and the way in which this is done is also described.

Self-test questions

1. What is the command to start the procedure for saving the program stored in the TI 99/4A on a cassette?
2. What are the BASIC words used for:
   (a) repetition
   (b) making a test and acting on the result,
   and
   (c) giving data to a program while it is running?
3. Write short programs for the following:
   (a) to print your name 10 times
   (b) to enter a word and decide if it has more than 7 characters. If it has more than 7 characters, indicate that a long word was entered, otherwise print that it was a short one;
   (c) to accept different words entered at the keyboard and then print them out without either their first or last letter.

4. Explain in the way illustrated in Figure 3.1 the computations performed when the following programs are executed:

(a)
```
10 A$ = "ALGORITHMIC"
20 L = LEN (A$)
30 FOR I = 1 TO L
40 PRINT SEG$(A$,I,1)
50 NEXT I
```

(b)
```
10 A = 1
15 B = 1
20 PRINT A
25 PRINT B
30 FOR I = 1 TO 12
40 C = A + B
50 PRINT C
60 A = B
65 B = C
70 NEXT I
```

5. Write a program to accept a word, store it in A$ and then create in B$ the reverse of the word. This can be done by starting with a string of zero characters in B$, and then adding one character at a time from the right of A$. The program should print the reversed word and then decide if the original word is a palindrome, that is, if it reads the same forwards and backwards.

A typical dialogue from the program might be:

```
ENTER A WORD, PLEASE.
?MADAM
THE REVERSE OF MADAM IS MADAM
MADAM IS PALINDROMIC.
```

Chapter 4
Graphics

In computer terminology, graphics are pictures. Many of the programs written for the TI 99/4A that will be of lasting interest and value will make good use of graphics. This will particularly apply to educational programs (see Figure 4.1) and computer games, where the interest and compulsion of the programs often lies in the attractiveness of the graphics. Business programs can also be made more effective if they present information and results in pictorial as well as numerical form.

Obviously, a large degree of computation is necessary in any reasonably complex program whatever its application, but the results from that computation can be presented in one of three ways: by numbers, words or pictures. While it is necessary in some applications to have accurate numerical results, in many others the presentation of a screen full of numbers inevitably becomes rather dull sooner or later. To present information using words is better, but it is easier to read from a book than

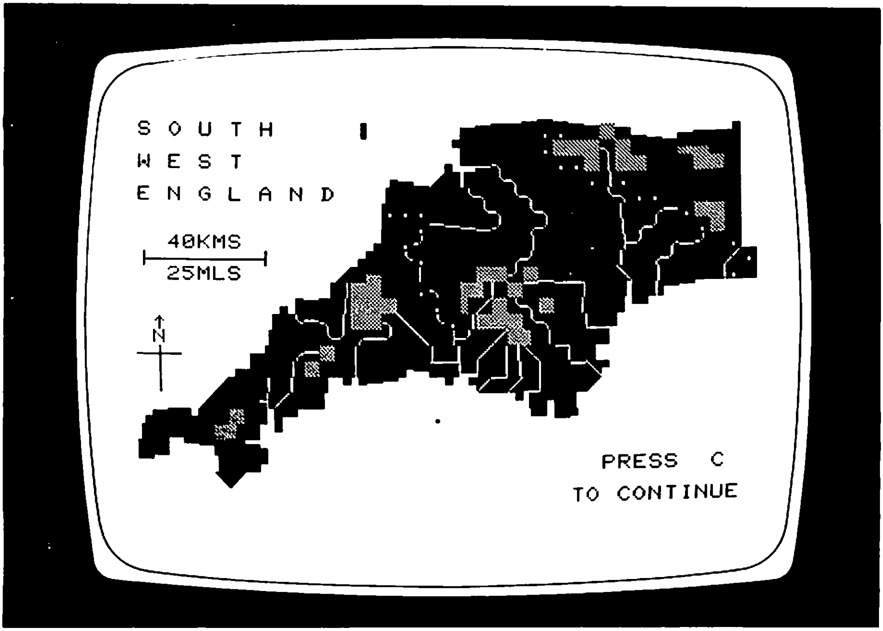


Figure 4.1 Display produced by 'Geography Tester' program.

from a video screen! Anyway, as everybody knows, a picture is worth a thousand words, and pictorial presentations are much more natural and informative than their alternatives. In this chapter we look at the TI 99/4A's graphics capabilities.

The TI 99/4A microcomputer has the ability to produce both graphics and colours on the screen. Furthermore, the graphics may be of either 'low resolution' or 'high resolution'. In this context, resolution refers to the size of the individual graphics elements that appear on the screen. Low resolution graphics thus refers to graphics where the elements are fairly large and the consequent clarity and definition of pictures is relatively poor. High resolution graphics produce much smaller graphics elements, and the consequent clarity and definition of the pictures produced is that much higher.

The TI 99/4A differs from many other home computers, however, in that the method of producing both high and low level graphics is exactly the same. Put very simply, the computer contains a small subprogram that is called CHAR. This subprogram can be invoked by the command: CALL CHAR. The full structure of the command is:

```
CALL CHAR (character code, character definition)
```

The character code is the ASCII code number for any standard ASCII symbol, plus some graphics characters. To see what codes produce which symbols, try the following program:

```
100 CALL CLEAR
110 FOR I = 1 TO 255
120 PRINT I;"(SPC)IS THE CODE FOR(SPC)";CHR$(I)
130 FOR J = 1 TO 100
140 NEXT J
150 NEXT I
```

A number of the early codes appear as blanks, as do many of the later codes. Code 32, however, is actually a space. See Figure 4.2 for a full list of the standard ASCII codes. What the CALL CHAR subprogram does is to redefine that character code with the character definition that follows.

For the purpose of defining the new character, CALL CHAR treats each character, as it appears on the screen, as a matrix of 8 x 8 squares. (See Figure 4.3.) You can thus redefine any existing character to any pattern you like made up of a square of 64 much smaller squares. This is why we say that the TI 99/4A is able to produce both high and low resolution graphics. If you treat the entire matrix as a single square that you either turn 'on' or 'off', or as four smaller squares, each 4 x 4, then you have a basic low resolution graphics set. Using the former definition, you will

The defined characters on the TI 99/4A Computer are the standard ASCII characters for codes 32 through 127. The following chart lists these characters and their codes.

ASCII CODE	CHARACTER		ASCII CODE	CHARACTER		ASCII CODE	CHARACTER	
32		(space)	65	A		97	a	
33	!	(exclamation point)	66	B		98	b	
34	"	(quote)	67	C		99	c	
35	#	(number or pound sign)	68	D		100	d	
36	$	(dollar)	69	E		101	e	
37	%	(percent)	70	F		102	f	
38	&	(ampersand)	71	G		103	g	
39	'	(apostrophe)	72	H		104	h	
40	(	(open parenthesis)	73	I		105	i	
41	)	(close parenthesis)	74	J		106	j	
42	*	(asterisk)	75	K		107	k	
43	+	(plus)	76	L		108	l	
44	,	(comma)	77	M		109	m	
45	−	(minus)	78	N		110	n	
46	.	(period)	79	O		110	o	
47	/	(slant)	80	P		112	p	
48	0		81	Q		113	q	
49	1		82	R		114	r	
50	2		83	S		115	s	
51	3		84	T		116	t	
52	4		85	U		117	u	
53	5		86	V		118	v	
54	6		87	W		119	w	
55	7		88	X		120	x	
56	8		89	Y		121	y	
57	9		90	Z		122	z	
58	:	(colon)	91	[	(open bracket)	123	{	(left brace)
59	;	(semicolon)	92	\	(reverse slant)	124	¦	
60	<	(less than)	93	]	(close bracket)	125	}	(right brace)
61	=	(equals)	94	^	(exponentiation)	126	~	(tilde)
62	>	(greater than)	95	_	(line)	127	DEL	(appears on screen as a blank.)
63	?	(question mark)	96	`	(grave)			
64	@	(at sign)						

These character codes are grouped into twelve sets for use in color graphics programs.

Set #	Character Codes	Set #	Character Codes	Set #	Character Codes
1	32-39	5	64-71	9	96-103
2	40-47	6	72-79	10	104-111
3	48-55	7	80-87	11	112-119
4	56-63	8	88-95	12	120-127

Two additional characters are predefined on the TI 99/4A Computer. The cursor is assigned to ASCII code 30, and the edge character is assigned to code 31.

Figure 4.2 The standard ASCII codes.

have a screen grid for drawing pictures of the standard 32 columns by 24 lines (768 separate positions). If you use the latter definition, then the screen grid becomes 64 x 48 (3072 positions). We can call both of these low resolution. But if you define each individual small square on the 8 x 8 matrix, then the total number of individual points that you can turn 'on' on the screen becomes 24 x 32 x 64, or 49,152 positions, which is better defined as high resolution.

All characters on the TI 99/4A are created by turning 'on' some of the dots in this 8 x 8 matrix and leaving others 'off'. The space character (32) is a matrix with all the dots turned 'off'; while the block cursor is a character with all the dots turned 'on'. The standard characters, 'A', 'B' 'C' etcetera, are automatically set by the computer, but you can create new characters by first specifying the character code that you wish to redefine, and then telling the computer which dots to turn 'on' and which dots to turn 'off'. To do this, imagine the 64 dot square as being composed of eight rows of 2 blocks of four dots. Always think of these blocks in a strict sequence of sixteen four dot blocks reading from left to right and top to bottom. If you understand binary and hexadecimal notation, think of each four dot block as a binary code, and then express it in its hexadecimal code. If you don't understand these codes, don't worry; just refer to Figure 4.4. So, in order to define a new character with the CALL CHAR subprogram, simply specify the character code, and then define the character as a string of 16 separate codes. In this way, the command:

```
CALL CHAR (33,"383810387CBA386C")
```

will redefine ASCII character 33 into the character shown, much enlarged, in Figure 4.5

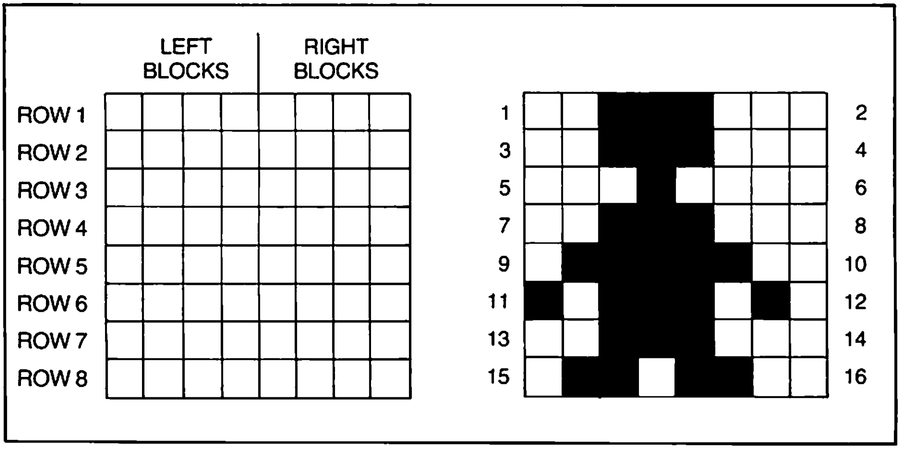


Figure 4.3 (a) The CALL CHAR character matrix. (b) A redefined character using CALL CHAR.

Blocks	Binary Code (0=Off; 1=On)	Hexadecimal Code
□□□□	0000	0
□□□■	0001	1
□□■□	0010	2
□□■■	0011	3
□■□□	0100	4
□■□■	0101	5
□■■□	0110	6
□■■■	0111	7
■□□□	1000	8
■□□■	1001	9
■□■□	1010	A
■□■■	1011	B
■■□□	1100	C
■■□■	1101	D
■■■□	1110	E
■■■■	1111	F

Note: The hexadecimal codes A, B, C, D, E and F must be entered from the keyboard as upper-case characters.

Figure 4.4 The CALL CHAR character definition codes.

To describe the dot pattern pictured below you would code this string for CALL CHAR:

"1898FF3D3C3CE404"

	LEFT BLOCKS	RIGHT BLOCKS	BLOCK CODES
ROW 1	□□□■	■□□□	18
ROW 2	■□□■	■□□□	98
ROW 3	■■■■	■■■■	FF
ROW 4	□□■■	■■□■	3D
ROW 5	□□■■	■■□□	3C
ROW 6	□□■■	■■□□	3C
ROW 7	■■■□	□■□□	E4
ROW 8	□□□□	□■□□	04

Figure 4.5 A redefined character using CALL CHAR.

We recommend that until you become more accomplished at programming the TI 99/4A, you concentrate on using the graphics facility at a low resolution level. On the face of it, it might seen that only a limited range of pictures could be generated within this format, but, as many existing programs have shown, and a number of the programs in this chapter will also demonstrate, displays of surprising complexity can still be produced. Some patience and ingenuity may be required to produce them, but a little knowledge and some effort are really all that is needed to start. Many people find that investigating and using the graphics facilities of the TI 99/4A are among its most interesting aspects. The inclusion of good graphic effects has certainly been a major reason for the success of many of the better programs written for microcomputers, and will furthermore help to ensure that new programs become a source of lasting pleasure and usefulness.

The screen and memory

A number of microcomputers use the facility of a POKE command to push special symbols into particular positions on the screen in order to build up pictures. This is NOT the method used by the TI 99/4A, and we mention it only because nearly everybody has heard of 'memory-mapped' screens and the POKE command.

In fact, the principle of graphics on the TI 99/4A is very similar. Imagine the screen as a grid of small squares. Each square can be defined as an X,Y co-ordinate. X is the number of the square from top to bottom (that is, the column); and Y is the number from left to right (that is, the row). The top left square is always 1,1.

The total number of squares actually depends on the graphics resolution you use via CALL CHAR. But under normal circumstances, it is exactly the same as the size of the text screen; that is 24 lines by 32 columns, see Figure 4.6. Thus in the low resolution grid size, the first row of squares will be numbered 1 to 32 from left to right, and can be accessed by 1,1; 1,2; 1,3; to 1,32. By the same principle, the bottom row of squares can be accessed by the co-ordinates 24,1; 24,2; 24,3; to 24,32. If you are using 'high resolution' graphics you will use exactly the same grid size and references, but you will have already defined each individual graphics character in considerable detail. Note that on some screens you cannot access all of the grid squares, notably the first one and the last two columns. This problem does not affect the way in which the squares are numbered, but you may therefore like to consider your grid as having columns numbered from 3 to 30.

We have already seen how to define a particular graphics character. All we need now is to know how to place that character at a specified position on the screen. The two commands (each of which is another subprogram) are CALL HCHAR and CALL VCHAR. In full, the commands are:

```
CALL HCHAR } (row number, column number,
CALL VCHAR } character code, number of repetitions)
```

'Row number' and 'column number' are the X,Y co-ordinates for the screen grid that we have just looked at. The 'character code' is that same ASCII code we have also examined already. Finally, the 'number of repetitions' is an optional extra that we may use to repeat the character a specified number of times in either a horizontal (HCHAR), or vertical (VCHAR) line.

We can enhance the use of these commands by adding an extra subprogram command: CALL COLOR. The full command is:

```
CALL COLOR  (character set number, foreground colour,
             background colour)
```

Every character is made up of two colours on the screen. The foreground colour is the colour of the dots that are turned 'on' in the 8 x 8 matrix. The

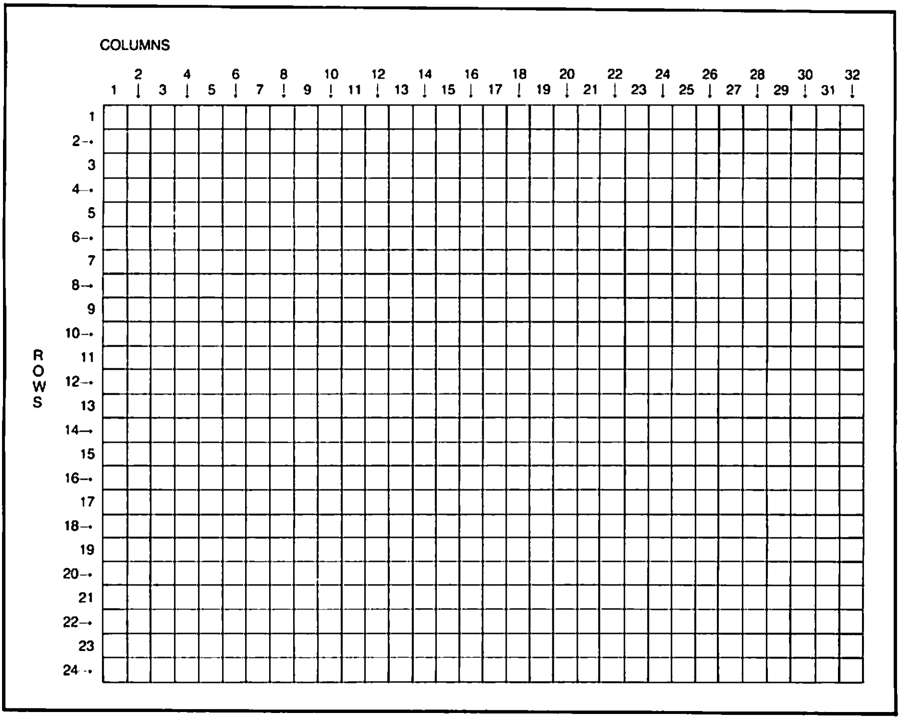


Figure 4.6 The CALL CHAR character grid.

background colour is the colour of the remaining dots that are turned ‘off’. You can use this command to specify the foreground and background colour of any graphics or other character you care to use. See Figure 4.7 for the colours available and the codes to use. Note that if you do not use the CALL COLOR subprogram, the standard default setting that will be used automatically by the computer is Black foreground (code 2) and Transparent background (code 1). When the background is transparent, those dots are shown in the same colour as the screen itself, and they thus become indistinguishable from rest of the screen. You can change the screen colour by using:

```
CALL SCREEN (colour code)
```

The colour code is a numeric expression with a value of between 1 and 16 (refer to Figure 4.7).

To test these commands, enter and run the following program. Try to work out exactly which command causes what effect on the screen:

```
10 CALL CLEAR
20 CALL COLOR(12,3,9)
30 K = INT(RND★23)+1
```

Colour Code	Colour
1	Transparent
2	Black
3	Medium Green
4	Light Green
5	Dark Blue
6	Light Blue
7	Dark Red
8	Cyan
9	Medium Red
10	Light Red
11	Dark Yellow
12	Light Yellow
13	Dark Green
14	Magenta
15	Grey
16	White

Figure 4.7 The colours and their codes.

```
40 CALL HCHAR(K,3,127,28)
50 L = INT(RND★27)+3
60 CALL VCHAR(1,L,127,24)
70 CALL HCHAR(K+1,3,32,28)
80 CALL VCHAR (1,L+1,32,24)
90 GOTO 20
```

Producing a drawing

To demonstrate the production of a picture, try a short program that draws the infamous space invader. We will be using a definition that really belongs to low resolution graphics since we shall build the picture out of four separate character spaces.

```
 10 CALL CLEAR
 20 A$ = "03030F0F3C3C3F3F"
 30 B$ = "F0F0FCFCCFCFFFFF"
 40 C$ = "0F0F03030C0C3030"
 50 D$ = "3C3CF0F00C0C0303"
 60 CALL CHAR(33,A$)
 70 CALL CHAR(34,B$)
 80 CALL CHAR(35,C$)
 90 CALL CHAR(36,D$)
100 CALL COLOR(1,7,12)
110 X = 5
120 Y = 7
130 CALL HCHAR(X,Y,33)
140 CALL HCHAR(X,Y+1,34)
150 CALL HCHAR(X+1,Y,35)
160 CALL HCHAR(X+1,Y+1,36)
170 GOTO 170
```

A 'space invader' is an artificial image in the sense that it originally took its form from the generally available graphics characters rather than from an inherent shape of its own that is later modelled, or approximated, by graphics characters. In this section a procedure for sketching a shape on the screen is described. It demonstrates that recognisable sketches can be produced, while at the same time showing that the limited definition of low resolution displays can cause problems regarding the accuracy of the sketch.

Suppose we want to draw the butterfly shown in Figure 4.8(a) on the screen. To do this, draw a square grid over it as shown in Figure 4.8(b), and then, for each square of the grid in turn find the graphics character most closely approximating to it. In fact, to demonstrate the principle, we shall simply use two graphics characters: a whole block turned on and a whole block turned off. The result of this will be something like that shown in Figure 4.8(c), while the outline of the butterfly as it will appear on the screen is shown in Figure 4.8(d). Finally, find the codes for the graphics characters and write a program to print them in the right place on the screen.

We will use a basic programming technique that you should try to understand and use as soon as possible. This is the use of the READ and DATA commands. The names are fairly self-explanatory; a READ command reads data from a DATA list. The first READ command executed in a program will read the first data item from the DATA list, the second takes the second item, and so on. By putting the READ command into a FOR . . . NEXT loop, you can read through the entire DATA lists quite simply. The information required to position the butterfly is stored as data in lines 300 to 360. That data is then read into the program proper by the READ commands in lines 90, 130 and 160. The structure of the program is therefore to store all the screen locations in lines 300 to 360, and all the graphics definitions in lines 10 and 20. Lines 30 and 40 redefine ASCII codes 33 and 34 as either a block on or a block off. Once these parameters are established, the program can then set up loops to READ the position specifications and place the characters via the HCHAR command.

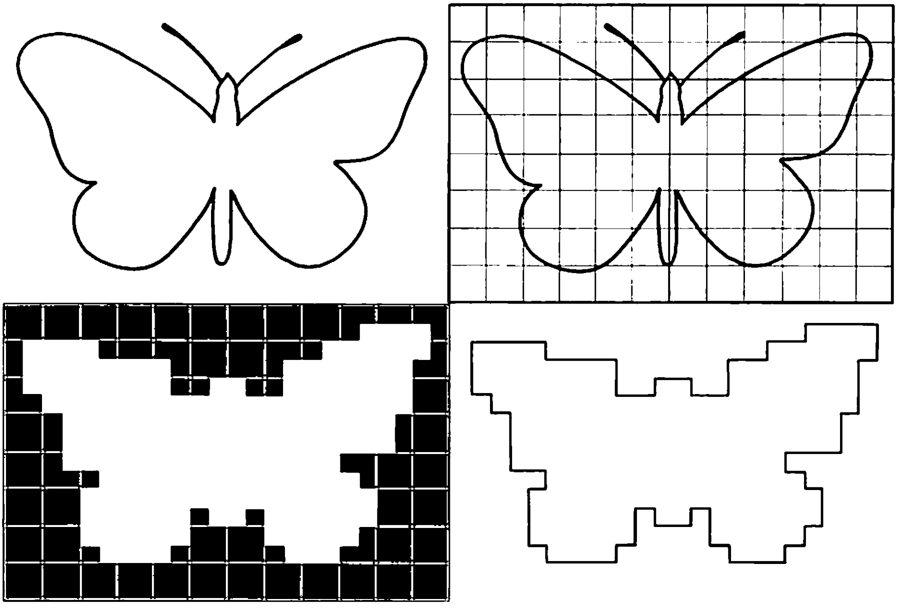


Figure 4.8 (a) Butterfly. (b) Butterfly with grid. (c) Butterfly composed of graphics characters. (d) Outline of image plotted on screen.

```
  5 CALL CLEAR
 10 A$ = "FFFFFFFFFFFFFFFF"
 20 B$ = "0000000000000000"
 30 CALL CHAR(34,A$)
 40 CALL CHAR(33,B$)
 50 CALL COLOR(1,12,2)
 60 CALL CLEAR
 70 Y = 5
 80 FOR I = 1 TO 16
 90 READ N
100 N = INT(N/2)
110 X1 = 1
120 FOR J = 1 TO N
130 READ X2
140 CALL HCHAR(Y,X1+3,33,X2-X1)
150 X1 = X2
160 READ X2
170 CALL HCHAR(Y,X1+3,34,X2-X1)
180 X1 = X2
190 NEXT J
200 X1 = X2
210 CALL HCHAR(Y,X1+3,33,15-X1)
220 Y = Y+1
230 NEXT I
240 GOTO 240
300 DATA 3,10,12,5,3,3,10,13
310 DATA 5,1,5,10,14,7,1,6,7,8,9,14
320 DATA 3,1,14,3,1,14,3,1,14
330 DATA 3,1,14,3,2,13,3,3,12
340 DATA 7,1,6,7,8,9,14,7,1,6,7,8,9,14
350 DATA 7,1,6,7,8,9,14,5,1,6,9,14
360 DATA 5,3,5,10,13,5,3,5,11,13
```

The problems of resolution can be tackled in a number of ways. The simplest is to stand further away from the screen, letting your eye and brain integrate and resolve the image as its fine detail becomes less clear. A more active measure would be to switch to high resolution graphics that would enable the use of a much larger number of grid squares. The positioning of the grid is also important, since the details that are vital for recognition should be captured as accurately as possible. Finally, a little artistic licence in the design of the displayed image may also help considerably.

Screen patterns

The screen can be filled with a particular symbol by the program:

```
10 CALL CLEAR
20 FOR I = 1 TO 24
30 CALL HCHAR(I,1,35,32)
40 NEXT I
50 GOTO 50
```

To change the character, simply change the third value held in parentheses; that is, 35.

When the character at each screen position is generated by a systematic method, patterns that can be both informative and aesthetically pleasing can be produced. A general scheme that can be used to give a wide variety of interesting patterns involves the three stages of computation, classification, and representation. A value is computed for each position on the screen using its row and column number. The value is then classified by assigning it to one of a number of classes. Each class is represented by a particular character. In this way a character can be obtained for, and plotted in, each screen position. The process is, essentially, that used to make a coloured contour map where the height of each point is measured

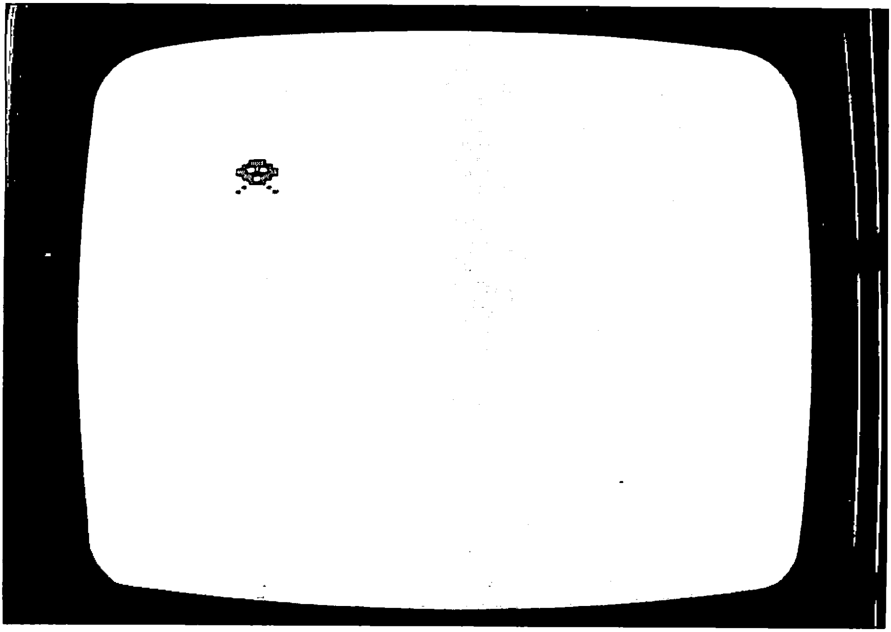


Figure 4.9 A space invader.

(computed), classified into the appropriate height interval, and then represented on the map by the colour assigned to that height interval. A general program scheme for generating screen patterns of this nature is given in Figure 4.10, and this can be refined to give a program such as the following:

```
10 CALL CLEAR
20 FOR I = 1 TO 4
30 READ P(I)
40 NEXT I
50 DATA 35,37,33,32
60 FOR R = 1 TO 24
```

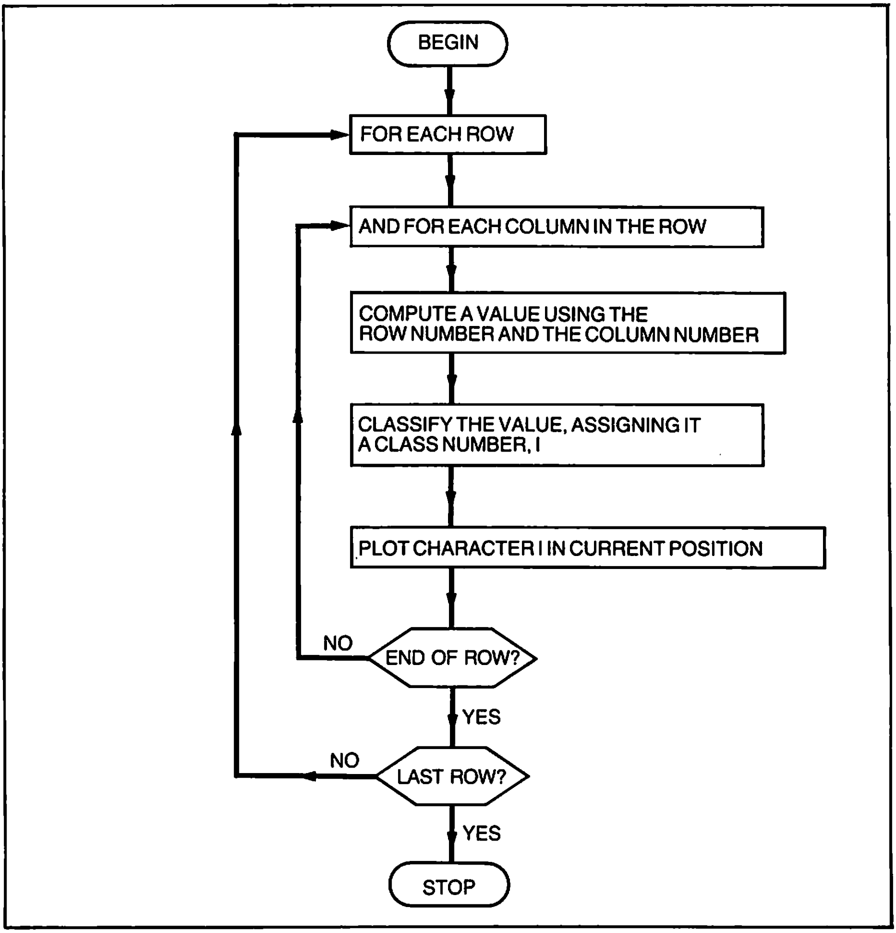

BEGIN
FOR EACH ROW
AND FOR EACH COLUMN IN THE ROW
COMPUTE A VALUE USING THE ROW NUMBER AND THE COLUMN NUMBER
CLASSIFY THE VALUE, ASSIGNING IT A CLASS NUMBER, I
PLOT CHARACTER I IN CURRENT POSITION
NO
END OF ROW?
YES
NO
LAST ROW?
YES
STOP

Figure 4.10 Program scheme for screen patterns.

```
 70 FOR C = 1 TO 32
 80 H = C★C + R★R
 90 IF H > 80 THEN 120
100 I = 1
110 GOTO 190
120 IF H = > 400 THEN 150
130 I = 2
140 GOTO 190
150 IF H > 800 THEN 180
160 I = 3
170 GOTO 190
180 I = 4
190 CALL HCHAR(R,C,P(I))
200 NEXT C
210 NEXT R
220 GOTO 220
```

If you follow the program closely, you will see that it keeps very close to the program scheme in Figure 4.10. (Figure 4.11 shows the result of running this program.) Line 60 is for each row, while line 70 is for each column. Line 80 computes a value, while lines 90 to 180 classify the values. Line

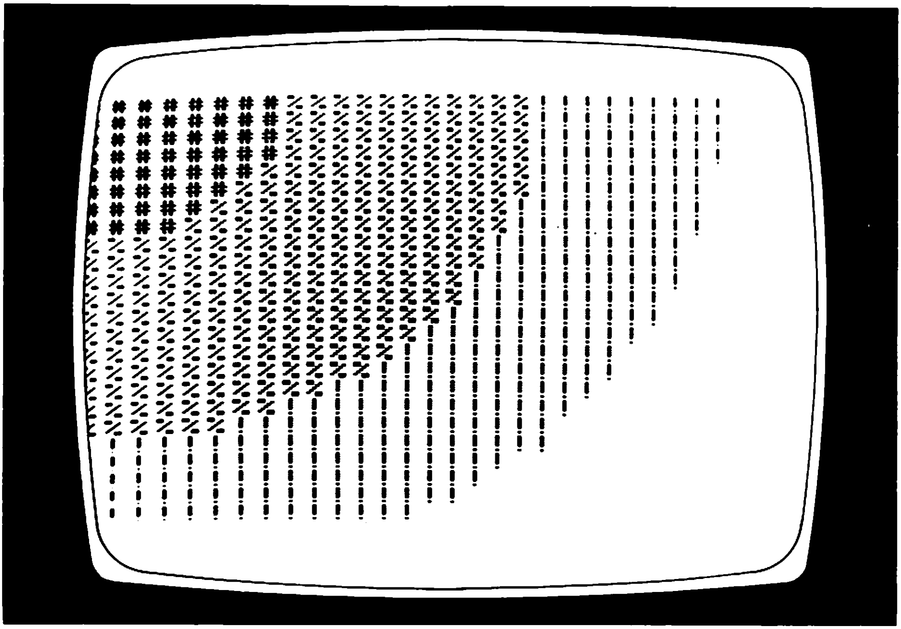


Figure 4.11 Screen pattern.

190 then plots the relevant class code on the individual points of the screen. (To see how colours can be used to plot a coloured contour map, you will need to redefine a character or characters as a solid block or blocks with CALL CHAR, and then include different colours with CALL COLOR. Try and do this for yourself.)

A second program following the same program scheme is:

```
 10 CALL CLEAR
 20 DIM P(10)
 30 FOR I = 1 TO 10
 40 READ P(I)
 50 NEXT I
 60 DATA 35,37,33,38,39,40,41,42,43,32
 70 FOR R = 1 TO 24
 80 FOR C = 1 TO 32
 90 H = (R★C)↑(1/3)
100 I = INT(H)
110 CALL HCHAR(R,C,P(I))
120 NEXT C
130 NEXT R
140 GOTO 140
```

Here there are 10 intervals and plotting symbols. Line 90 computes the value (R★C raised to the power of 1/3; that is, the cube root of R★C), while line 100 classifies the value by finding its whole part (the INT command). To change the appearance of the pattern, try changing the values of the symbols being used.

A wide range of patterns can be produced by using this method. In general, a distinct pattern results from each choice of computation, classification and set of plotting symbols chosen to represent the classes. Classification can be achieved in many ways other than by dividing a range of values into intervals; for example, the number in the first place after the decimal point in the computed value can be used to give the class number. The selection of plotting characters is vital to the presentation of effective patterns. The characters chosen for the last two programs are intended to accentuate the transition from one class to another, but other characters may well prove more effective or attractive.

Movement

Once static displays can be produced, it seems natural to progress to the generation of moving displays. The programs presented in this section make it possible for the user to control the movement of a shape on the screen. Besides being fascinating in itself such programs illustrate the techniques used in many games programs.

It is worth mentioning that the TI 99/4A has the facility to use joysticks to control movement on the screen, but since this is a more advanced form of programming and it is quite likely that you did not buy the joysticks when you first bought the computer, we shall merely give an example of how to adapt our earlier space invader program into one that will provide a moving space invader via one of the joysticks. If you do have the joysticks, then you will be able to experiment with and expand the program into some form of game. If not, still try to work out how the program works. It makes the same space invader that we drew earlier move in any of the eight major compass directions. A refinement is needed, however, to prevent the moving shape from leaving a trail behind itself. Movement is simulated by redrawing the entire shape by one position to the left or right, etcetera. If it moves to the right, however, it will leave its leftmost characters where they were on the screen. One method of avoiding this is for the shape to have a surround of spaces so that the part left behind is always blank. In the following program, which you will see is an expansion of our earlier space invader program, lines 280 to 310 are added to prevent the trailing effect.

```
 10 CALL CLEAR
 20 A$ = "03030F0F3C3C3F3F"
 30 B$ = "F0F0FCFCCFCFFFFF"
 40 C$ = "0F0F03030C0C3030"
 50 D$ = "3C3CF0F00C0C0303"
 60 CALL CHAR(33,A$)
 70 CALL CHAR(34,B$)
 80 CALL CHAR(35,C$)
 90 CALL CHAR(36,D$)
100 CALL COLOR(1,7,12)
110 X = 10
120 Y = 10
130 CALL JOYST(1,T,S)
140 X = X-S/4
150 IF X > 2 THEN 170
160 X = 2
170 IF X < 20 THEN 190
180 X = 20
190 Y = Y + T/4
200 IF Y > 2 THEN 220
210 Y = 2
220 IF Y < 29 THEN 240
230 Y = 29
```

```
240 CALL HCHAR(X,Y,33)
250 CALL HCHAR(X,Y+1,34)
260 CALL HCHAR(X+1,Y,35)
270 CALL HCHAR(X+1,Y+1,36)
280 CALL HCHAR(X−1,Y−1,32,4)
290 CALL HCHAR(X+2,Y−1,32,4)
300 CALL VCHAR(X−1,Y−1,32,4)
310 CALL VCHAR(X−1,Y+2,32,4)
320 GOTO 130
```

Notice line 130 and the JOYST command. This command takes the value of −4, 0, or 4 depending on the position of the joystick. For further information on this, refer to the Texas Instruments Reference Manual. Lines 150 to 230 prevent the invader from wandering off the edge of the screen. Without this routine, the program would fail whenever the invader goes off the edge.

Animation

Displaying slightly different still pictures at a sufficiently high rate produces the illusion of continuous movement. All moving-picture systems including films and television rely on this effect; an effect that depends on several human characteristics including the persistence of vision. The program presented in this section attempts to produce a mobile display by plotting successive static images in precisely the same way as a moving picture is produced by showing successive static images at a fast enough rate.

The following program produces an animated display of a flying butterfly: the successive frames catch different positions of the butterfly's wing when in flight. The sequence from which the frames are derived is shown in Figure 4.12. The first frame, with the wings fully extended, is the image produced earlier. The other frames are obtained in the same way as the first. In the program, the codes for the three frames are read in first, and then the frames are plotted repeatedly in the sequence 1,2,3,2. The flow chart of the program is given in Figure 4.13.

The program is:

```
 5 CALL CLEAR
10 A$ = "FFFFFFFFFFFFFFFF"
20 B$ = "0000000000000000"
30 CALL CHAR(34,A$)
40 CALL CHAR(33,B$)
50 CALL COLOR(1,12,2)
60 FOR K = 1 TO 4
```

```
 70 Y = 5
 80 FOR I =1 TO 16
 90 READ N
100 N = INT(N/2)
110 X1 = 1
120 FOR J = 1 TO N
130 READ X2
140 CALL HCHAR(Y,X1+3,33,X2-X1)
150 X1 = X2
160 READ X2
170 CALL HCHAR(Y,X1+3,34,X2-X1)
180 X1 = X2
190 NEXT J
200 X1 = X2
210 CALL HCHAR(Y,X1+3,33,15-X1)
220 Y = Y + 1
230 NEXT I
240 NEXT K
250 RESTORE
260 GOTO 60
270 REM   NUMBER ONE
300 DATA 3,10,12,5,3,5,10,13
```

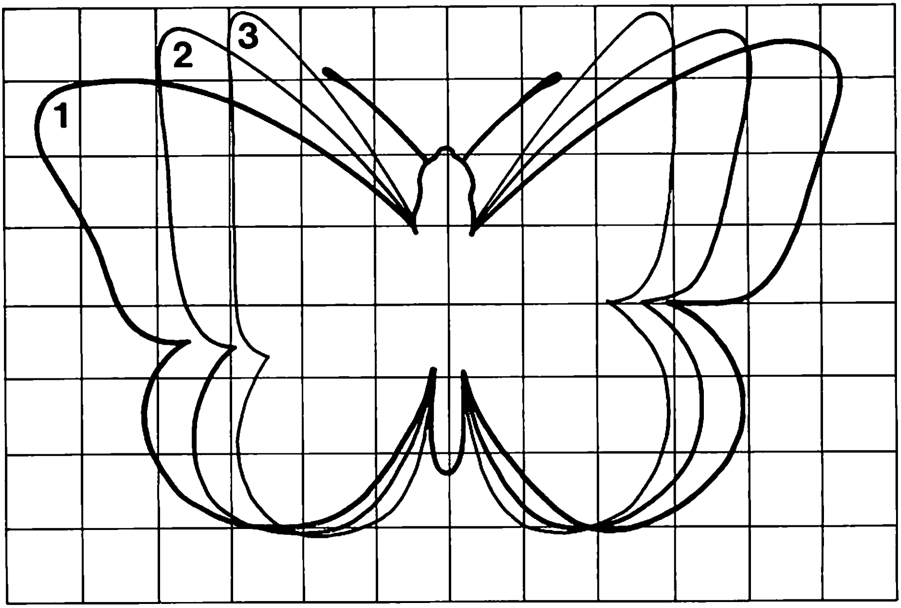


Figure 4.12 Frames 1, 2 and 3 for flying butterfly.

```
310 DATA 5,1,5,10,14,7,1,6,7,8,9,14
320 DATA 3,1,14,3,1,14,3,1,14
330 DATA 3,1,14,3,2,13,3,3,12
340 DATA 7,1,6,7,8,9,14,7,1,6,7,8,9,14
350 DATA 7,1,6,7,8,9,14,5,1,6,9,14
360 DATA 5,3,5,10,13,5,3,5,11,13
370 REM    NUMBER 2
380 DATA 5,4,5,10,11,5,3,5,10,12
390 DATA 5,2,5,10,13,7,2,6,7,8,9,13
400 DATA 3,2,13,3,2,13,3,2,13,3,2,13
410 DATA 3,3,12,3,4,12,7,2,6,7,8,9,13
420 DATA 5,2,6,9,13,5,2,6,9,13,5,2,6,9,13
430 DATA 5,3,5,10,12,5,3,5,11,12
440 REM    NUMBER 3
450 DATA 5,4,5,10,11,5,3,5,10,12
460 DATA 5,3,5,10,12,7,3,6,7,8,9,12
470 DATA 3,3,12,3,3,12,3,3,12,3,3,12,3,4,11,3,3,12
480 DATA 7,3,6,7,8,9,12,5,3,6,9,12,5,3,6,9,12
490 DATA 5,3,6,9,12,5,4,5,10,11,5,4,5,10,11
500 REM    NUMBER 2
510 DATA 5,4,5,10,11,5,3,5,10,12
520 DATA 5,2,5,10,13,7,2,6,7,8,9,13
530 DATA 3,2,13,3,2,13,3,2,13,3,2,13
```

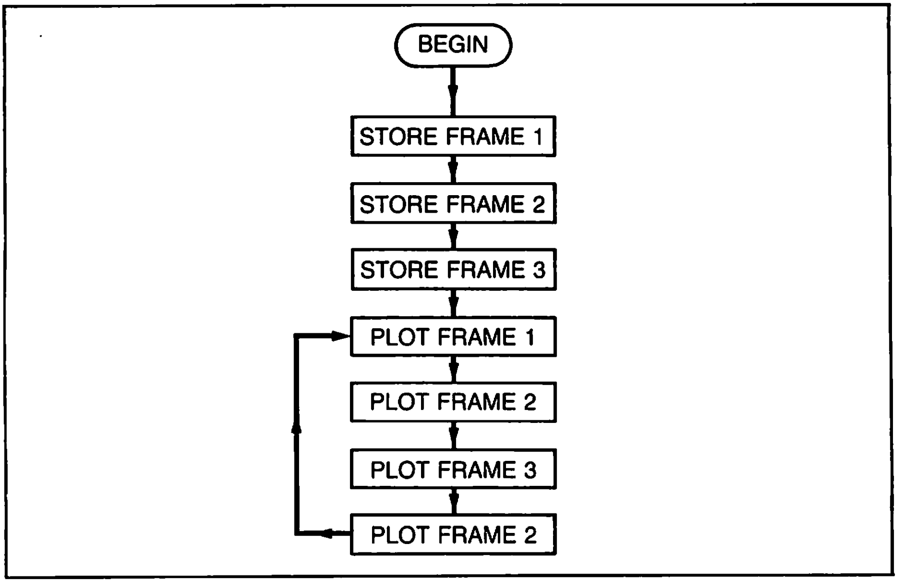


Figure 4.13 Flow chart for mobile display program.

```
540 DATA 3,3,12,3,4,12,7,2,6,7,8,9,13
550 DATA 5,2,6,9,13,5,2,6,9,13,5,2,6,9,13
560 DATA 5,3,5,10,12,5,3,5,11,12
```

This program shows how the simulation of animation can be achieved. In this particular instance it does not work quickly enough to be very effective. Obviously, the larger and more complicated the image, the longer it takes the computer to display the different frames, and the less effective the animation. One method that can be used to increase the frame speed sequence in animation is to plot only the changes necessary to convert one frame to the next rather than to plot entire frames all the time. You might also like to produce an animation of a much smaller display to see how much faster and more effective it becomes. Use the following program to produce the image of a small figure running from left to right across the screen:

```
100 CALL CLEAR
110 A$ = "383810387CBA386C"
120 B$ = "3838927C3838FE00"
130 C$ = "0000000000000000"
140 CALL CHAR(33,A$)
150 CALL CHAR(34,B$)
160 CALL CHAR(35,C$)
170 FOR I = 3 TO 27
180 CALL HCHAR(10,I,33)
190 CALL HCHAR(10,I,34)
200 CALL HCHAR(10,I,35)
210 NEXT I
220 GOTO 140
```

Because this figure is so much smaller and so much faster, we actually have to include some extra instructions to slow things down enough for us to see them. Add the four following lines:

```
185 FOR J = 1 TO 250
186 NEXT J
195 FOR J = 1 TO 250
196 NEXT J
```

These are delay loops. They do not actually do anything themselves, they simply instruct the computer to go round a loop a set number of times. Naturally, the larger the number of loops specified, the longer it takes, and consequently, the longer is the delay. In this instance, it is just long enough to let us see the little man jumping up and down as he runs from left to right.

Dynamic simulation

This section provides a dynamic simulation of a system that experiences random growth and decay. It displays a community that grows initially from a single cell. When it reaches a certain size it decays to a lower level and then fluctuates between those two levels. The display can be taken as a simulation of the growth of a town or of a community of insects, although budding town planners will already know that real towns do not grow randomly! The random element of the program is provided by a command RND that generates a pseudo random number.

```
  10 CALL CLEAR
  20 G = 1
  30 C = 1
  40 DIM A(12,16)
  50 T = 1
  60 A(5,6) = 1
  70 GOSUB 1000
  80 FOR I = 1 TO 12
  90 FOR J = 1 TO 16
 100 IF RND < 0.9 THEN 120
 110 A(I,J) = T
 120 NEXT I
 130 NEXT J
 140 C = 0
 150 FOR I = 1 TO 12
 160 FOR J = 1 TO 16
 170 IF A(I,J) = 0 THEN 190
 180 C = C+1
 190 NEXT J
 200 NEXT I
 210 G = G+1
 220 GOSUB 1000
 230 IF C < 99 THEN 250
 240 T = 0
 250 IF C > 27 THEN 270
 260 T = 1
 270 GOTO 80
1000 FOR I = 1 TO 12
1010 FOR J = 1 TO 16
1020 IF A(I,J) = 0 THEN 1050
1030 CALL HCHAR(I,J+5,42)
1040 GOTO 1060
```

```
1050 CALL HCHAR(I,J+5,32)
1060 NEXT J
1070 NEXT I
1080 RETURN
```

Chapter 5

Special features of the TI 99/4A

In this chapter, information about the TI 99/4A and some of its special features either not mentioned or mentioned only briefly in earlier chapters, is now gathered together to provide a basic reference chapter. It is not intended to be an exhaustive collection of data about the TI 99/4A, but it does include features that are of interest to the new user of the computer.

Specification of the TI 99/4A

Manufacturer:	Texas Instruments Inc
Microprocessor:	Texas Instruments 9900 family, 16-bit microprocessor, plus 256-byte scratchpad RAM.
Screen display:	32 characters by 24 lines. Note that on a number of screens, only columns 2 to 28 can be accessed.
Keyboard:	47 keys and one space-bar. The letters are laid out in typewriter style: QWERTY
Memory size:	The total combined memory can be up to 110K. 16K Bytes RAM supplied (expandable to 48K). 26K Bytes ROM supplied. A further maximum of 36K Bytes may be provided as external ROM by the Solid State Software command modules.
Language:	14K TI BASIC Interpreter built in. TI-Extended BASIC, LOGO, Pascal, and TMS 9900 Editor/Assembler available on Solid State Software command modules.
Graphics:	Individual characters may be redefined to any shape possible in an 8 x 8 matrix. The screen resolution is 192 x 256 dots, or 24 x 32 characters.

There are 16 foreground and background colours available:

colour code	colour
1	Transparent
2	Black
3	Medium Green
4	Light Green
5	Dark Blue
6	Light Blue
7	Dark Red
8	Cyan
9	Medium Red
10	Light Red
11	Dark Yellow
12	Light Yellow
13	Dark Green
14	Magenta
15	Grey
16	White

Peripherals:

The TI 99/4A will connect to a normal colour television (or black and white). It will also connect to a standard domestic cassette recorder.

Texas Instruments produces its own joysticks, known as remote controllers, that are wired together and plug into the port on the left-hand side of the console (facing from the front).

A TI 99/4A printer is also available. It is capable of printing both text and graphics characters. It can print 40 enlarged characters per line, 66 mixed characters per line, 80 normal characters, or 132 condensed characters. It is bidirectional, and can print at eighty characters per second.

The TI Solid State Speech synthesizer allows the addition of speech to the computer. It is entirely electronic. There are no taped voice recordings or any other traditional medium. Instead, a vocabulary of words and phrases is permanently stored on chips within the synthesizer. It has a resident vocabulary of almost 400 English words.

Expansions: The main method of expanding the system is by the Peripheral Expansion System, which allows you to start simply and then gradually build up a more sophisticated system by plugging in additional hardware cards. The unit measures 14 x 15 x 20 inches and has eight slots for the cards. One of them is used to connect the system to the computer console. The system also provides a space for the installation of a TI disk drive (up to two additional disk drives may be added externally).

The Memory Expansion board provides a further 32K bytes of Random Access Memory.

The Disk Memory System, consisting of a disk controller card and from 1 to 3 disk drive units, enables you to store information or data for later reuse. The disk drive is a single-sided single-density unit with the following features:

5.25 inches
up to 89K bytes capacity per disk
34 or 40 tracks
up to 127 files defined

Double-sided and double-sided/ double-density disk controllers and disk drives will be added to the range at a future date.

The RS232 Interface Card enables you to connect a wide range of accessory devices, including those from other manufacturers and other computers using ASCII protocol, to your own computer.

The P-Code card includes the UCSD-Pascal version IV.0 P-code interpreter which enables the user to run programs written for the TI 99/4A in Pascal. Programs written for other computers can also be run with little or no modification.

Software: Apart from the built-in BASIC interpreter, the TI 99/4A also has a built-in Internal Graphics Language interpreter, and a 4.4K byte monitor.

A large and growing library of applications software already exists for the TI 99/4A, covering education, games and business. Software supplied by Texas Instruments is usually on a Solid State Software command module which simply plugs into the port on the front of the console next to the keyboard. You do not need to load this software into the console's memory since it comes supplied with its own memory.

Third party software suppliers will usually provide their software on cassette or floppy disk.

Inside the TI 99/4A microcomputer

You must provide your own screen for the display, which could be a domestic television set (black and white or colour). The keyboard is similar to a typewriter differing only a few extra keys. Since these external features are familiar, most of this section is devoted to the inside of the computer.

Inside the TI 99/4A are the electronics for producing the specialised screen displays, the sound producing circuits, the memory, and of course all the logic for running a sophisticated computer. All of this is mounted on one medium sized printed circuit board, and a number of smaller boards.

A printed circuit board is the most convenient way to mount and interconnect the large number of components of the computer. It has copper tracks laid down on it to connect the mounts into which the various chips are inserted. The layout of the printed circuit board reveals the essential structure of the microcomputer.

Unlike some other microcomputers, the TI 99/4A does not have its power supply unit within the main casing. The TI 99/4A's power unit is a separate box that connects the computer to the domestic mains supply. It converts the 240 volts alternating supply from the mains to the voltage required by the computer components.

The memory available to the user, particularly for storing BASIC programs, is provided by 'random access' chips, or RAMs. The information stored in this type of memory can be accessed, and can be replaced by the program as required. When the computer is switched off, all the information stored in RAM is lost.

There are, of course, certain features of the TI 99/4A that are always required and which must not be replaced or lost when the computer is turned off. To give just two examples: BASIC must always be available, and the characters to be displayed on the screen should be able to be generated at any time. Such functions are provided by chips with information permanently stored in them. These chips are known as 'read only memories', or ROMs.

The sockets for connecting the TI 99/4A computer to other devices are at the edge of the printed circuit board and at the two sides and the back of the case. The socket for the Solid State Command modules is at the front.

Sound on the TI 99/4A computer

The TI 99/4A computer enjoys a growing reputation for the quality and range of its sound capabilities. Sound is, furthermore, very simple to produce.

Unlike many of its competitors, the TI 99/4A computer does not include an internal loudspeaker. Instead, it uses the loudspeaker of the television set to which it is attached, and thereby achieves a better quality and greater control over the volume than many other microcomputers.

The form of the SOUND command, which is really a subprogram, is as follows:

```
CALL SOUND (duration,frequency one,volume one,frequency two,
        volume two,frequency three,volume three,frequency four,
        volume four)
```

The duration specifies how long the tone is to last, the frequency specifies which tone actually plays, and the volume controls how loud the tone is. All three are numeric expressions in the following ranges:

duration:	1 to 4250, inclusive −1 to −4250, inclusive
frequency:	(Tone) 110 to 44733, inclusive (see Figure 5.1) (Noise) −1 to −8, inclusive
volume:	0 (loudest) to 30 (quietest), inclusive

To get some idea of the range of tones that this command can produce, enter the following program:

```
100 TONE = 110
110 FOR C = 1 TO 10
120 CALL SOUND(−500,TONE,1)
130 TONE = TONE+110
140 NEXT C
150 FOR C = 10 TO 1 STEP −1
160 CALL SOUND(−500,TONE,1)
170 TONE = TONE−110
180 NEXT C
190 FOR I = 110 TO 7700 STEP 110
```

```
200 CALL SOUND(−500,I,1)
210 PRINT I
220 NEXT I
230 END
```

The use of the SOUND command can be used with great effect in games with graphics – Arcade Space Invaders will probably be familiar to everyone. To demonstrate the concept of mixing sound and graphics, try the following program:

```
 10 CALL CLEAR
 20 A$ = “FFFFFFFFFFFFFFFF”
 30 FOR I = 0 TO 15
 40 CALL CHAR(32+I★8,A$)
 50 NEXT I
 60 M = INT(RND★ 16)+1
```

MUSICAL TONE FREQUENCIES

The following table gives frequencies (rounded to integers) of four octaves of the tempered scale (one half-step between notes). While this list does not represent the entire range of tones – or even of musical tones – it can be helpful for musical programming.

Frequency	Note	Frequency	Note
110	A	440	A (above middle C)
117	A#,B^{b}	466	A#,B^{b}
123	B	494	B
131	C (low C)	523	C (high C)
139	C#,D^{b}	554	C#,D^{b}
147	D	587	D
156	D#,E^{b}	622	D#,E^{b}
165	E	659	E
175	F	698	F
185	F#,G^{b}	740	F#,G^{b}
196	G	784	G
208	G#,B^{b}	831	G#,A^{b}
220	A (below middle C)	880	A (above high C)
220	A (below middle C)	880	A (above high C)
233	A#,B^{b}	932	A#,B^{b}
247	B	988	B
262	C (middle C)	1047	C
277	C#,D^{b}	1109	C#,D^{b}
294	D	1175	D
311	D#,E^{b}	1245	D#,E^{b}
330	E	1319	E
349	F	1397	F
370	F#,G^{b}	1480	F#,G^{b}
392	G	1568	G
415	G#,A^{b}	1661	G#,A^{b}
440	A (above middle C)	1760	A

Figure 5.1 Musical tone frequencies.

```
 70 I = INT(RND★26)+1
 80 J = INT(RND★32)+1
 90 CALL COLOR(M,M,2)
100 CALL HCHAR(I,J,32+(M−1)★8)
110 CALL SOUND(10,INT(RND★600)+200,10)
120 GOTO 60
```

This program demonstrates that you can mix graphics and sound to produce interesting effects (see Figure 5.2). Try and work out which lines produce the graphics (and how they do it), and which lines produce the sounds. Any similarity to some of the more modern pop music is purely (and randomly!) coincidental.

By exercising tight control over the production of these sounds, you can make the computer produce quite acceptable music. Look at the following example. The structure of the program is very simple. The frequency and duration of each individual note are stored as data in lines 110 to 130, and 140 to 160. The data is then read and assigned variable names by two FOR . . . NEXT loops in lines 20 to 70, and then 'played' by a single CALL SOUND command in a third FOR . . . NEXT loop. It is a good idea to write all the music in this basic method: as data strings that are then

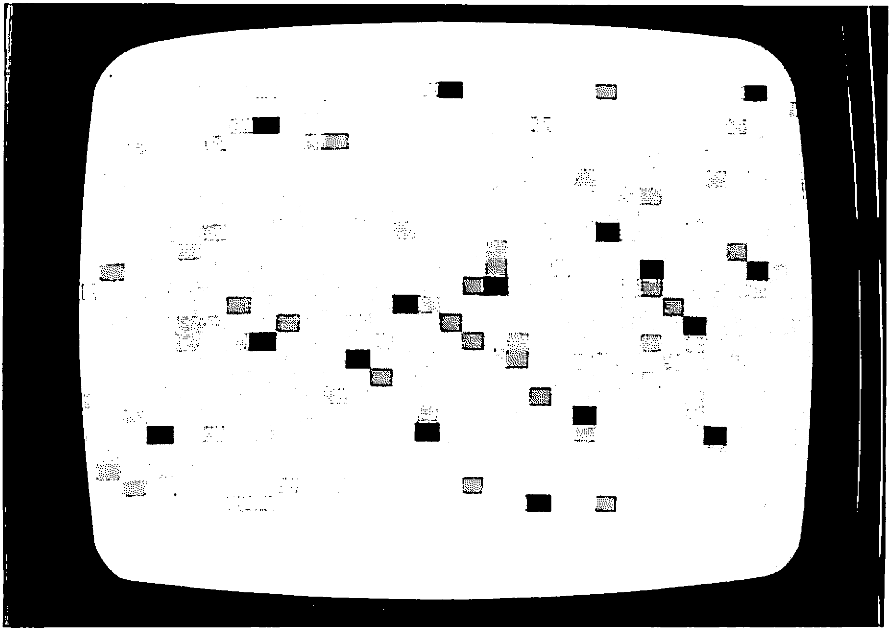


Figure 5.2 Display produced by the 'Random Sound and Colours' program.

read and played. In this way, repetitive pieces (choruses etc) can be played without retyping all the arguments. Although there are no repetitions in the following example, the principle is clear.

```
  5 CALL CLEAR
 10 DIM F(50),T(50)
 20 FOR I = 1 TO 45
 30 READ F(I)
 40 NEXT I
 50 FOR I = 1 TO 45
 60 READ T(I)
 70 NEXT I
 80 FOR I = 1 TO 45
 90 CALL SOUND(T(I)★190,F(I),15)
100 NEXT I
110 DATA 262,262,294,247,262,294,330,330,349,330,294,
    262,294,262,247,262
120 DATA 262,294,330,349,392,392,392,392,349,330,349,
    349,349,349,330,294
130 DATA 330,349,330,294,262,330,349,392,440,349,330,
    294,262
140 DATA 2,2,2,3,1,2,2,2,2,3,1,2,2,2,2,2
150 DATA 1,1,1,1,2,2,2,3,1,2,2,2,2,3,1,2
160 DATA 2,1,1,1,1,3,1,2,1,1,2,2,2,
```

Using the TI 99/4A as a timer

If you read other books on programming in BASIC, you may come across a command called TI or TI$. This command instructs the processor to display the current value of an internal counter that automatically starts whenever you turn on the computer. The timing of this counter is extremely accurate, and it is consequently often referred to as a clock. Computers that include this clock/counter can be readily used as an elaborate and very accurate form of timer. Unfortunately, at the time of writing, there is no such clock/counter in the TI 99/4A. Test this by typing in the command:

```
PRINT TI <ENTER>
```

The response, you will see, is to display:

```
  0
>
```

This section, however, will show that with a little imagination, the TI 99/4A can still be made to simulate a timer with a fair degree of accuracy. The routine itself could be incorporated quite effectively into a recipe program. Let us say that the recipe has given both the ingredients and the instructions, and that the mixture has to be cooked for a certain length of time. It could call the following as a subroutine (using GOSUB and RETURN) and act as a timer for the recipe. Enter the following program and try to work out what is happening. There are no new commands that we haven't already come across.

```
100 CALL CLEAR
110 PRINT "HOW LONG DO YOU WANT"
120 PRINT "TO SET THE TIMER FOR?"
130 PRINT
140 PRINT "ENTER THE NUMBER OF MINUTES";
150 INPUT A
160 CALL CLEAR
170 PRINT "COUNTING UP TO";A;"MINUTE/S"
180 FOR I=1 TO A
190 FOR J=1 TO 19675
200 NEXT J
210 CALL CLEAR
220 PRINT "COUNTING UP TO";A;"MINUTE/S"
230 PRINT
240 PRINT I;"MINUTE/S OF";A;"COUNTED"
250 CALL SOUND(10,262,15)
260 NEXT I
270 FOR I = 1 TO 4
280 FOR K = 110 TO 330 STEP 110
290 CALL SOUND(-500,K,15)
300 NEXT K
310 NEXT I
320 CALL CLEAR
330 PRINT "TIMES UP!"
340 PRINT
350 PRINT "DO YOU WANT TO RESET TIMER? Y/N";
360 INPUT B$
370 IF B$ = "Y" THEN GOTO 10
380 END
```

This program illustrates a number of the more simple features of BASIC that you have already come across: PRINT, FOR . . . NEXT, GOTO and CALL SOUND. But it also demonstrates an interesting use of the FOR . . . NEXT loop that is particularly useful when programming the TI 99/4A:

that is, as a 'delay loop'. Notice the format of the loop:

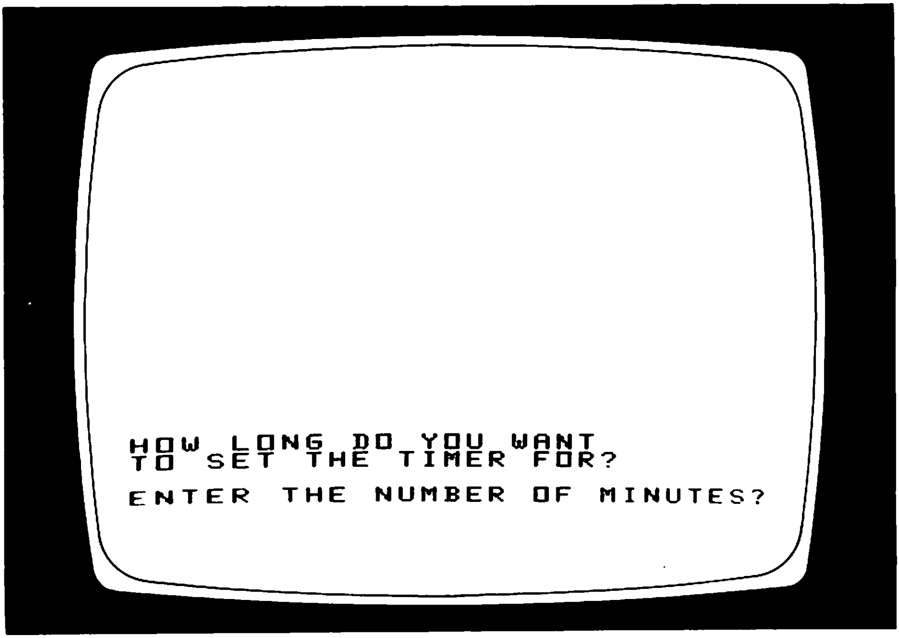
```
FOR I = 1 TO n
NEXT I
```

There is no separate command between the FOR and the NEXT part of the instruction. In other words, this construction instructs the computer to do nothing but go round in circles for a specified number of times. By varying the number of loops you can specify the length of time the computer takes to complete the command, and hence the duration of the delay introduced into the program.

Notice that our delay loop (which is the 'timer') at line 190 is for 1 to 19675 in steps of +1. Experimentation has shown that it takes the TI 99/4A almost exactly one minute to complete this number of loops. If you find that it is in fact only 59.9 seconds, try adding a few more loops: conversely, reduce the number to, say, 19650 if you find the TI 99/4A is taking too long.

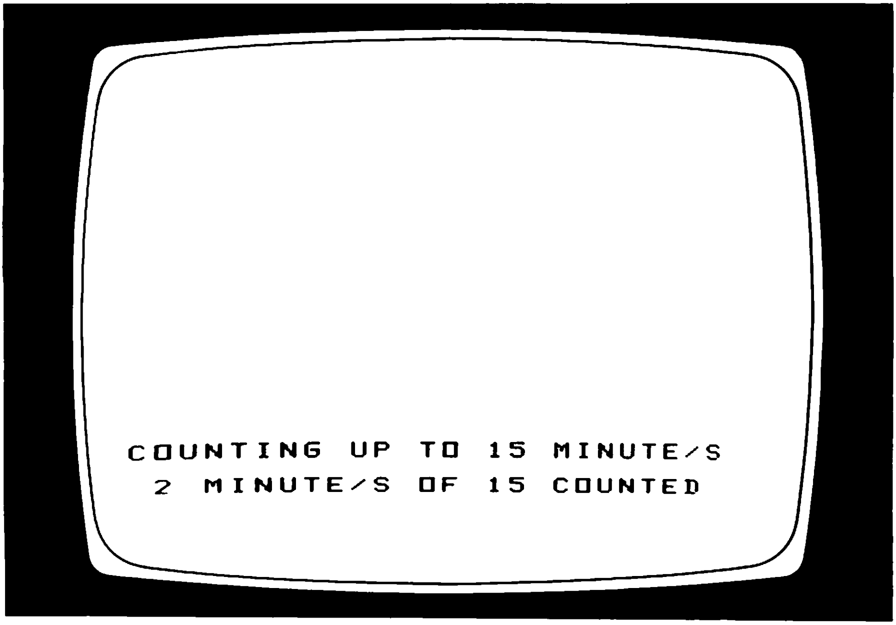
The timer also shows a feature known as 'nested loops'. The first loop is FOR I = 1 TO A. Now, A, as you will see at lines 140 and 150, is a variable input by the user to specify the duration of the timer in minutes. The FOR J loop is thus repeated as many times as the FOR I loop specifies.

Lines 240 and 250 provide a simple counter and buzzer to show the



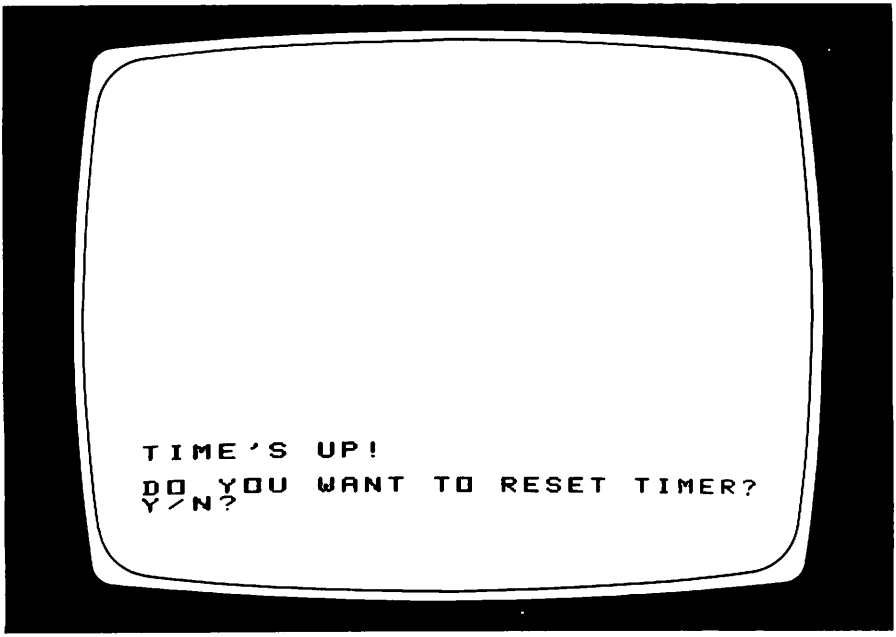
```
HOW LONG DO YOU WANT
TO SET THE TIMER FOR?
ENTER THE NUMBER OF MINUTES?
```

Figure 5.3 Display produced by the 'Timer' program.



```
COUNTING UP TO 15 MINUTE/S
 2 MINUTE/S OF 15 COUNTED
```

Figure 5.4 Display produced by the 'Timer' program.



```
TIME'S UP!
DO YOU WANT TO RESET TIMER?
Y/N?
```

Figure 5.5 Display produced by the 'Timer' program.

passage of time. Lines 270 to 310 provide the alarm that is activated as soon as the nested FOR . . . NEXT timer is complete, while the rest of the program provides the opportunity to reset the timer or exit from the program.

Speech synthesis

We won't say much about the Texas Instruments Speech Synthesizer, since its use depends upon having a Speech Synthesizer hardware unit and a software command unit that has speech capabilities. Possession of these items really takes the scope of the computer out of the limits set for a beginner's book such as this. Nevertheless, speech is such an exciting addition to modern computing that we include below a small spelling test program that makes the computer ask you how to spell certain words. If you have a speech synthesizer, then you can take this program and adapt and expand it into a sophisticated spelling test. If you don't have a synthesizer, it is still worth trying to work out what the program does.

```
  5 CALL CLEAR
 10 RANDOMIZE
 20 FOR I = 1 TO 10
 30 READ X$(I)
 40 NEXT I
 50 I = INT(RND★10+1)
 60 CALL SAY(“CAN YOU SPELL”)
 70 CALL SAY(X$(I))
 80 CALL KEY(3,A,C)
 90 IF C<1 THEN 80
100 A$ = CHR$(A)
110 PRINT A$
120 CALL SAY(A$)
130 P$ = P$&A$
140 IF P$ = X$(I) THEN 170
150 IF LEN(P$) > LEN(X$(I)) THEN 200
160 GOTO 80
170 CALL SAY(“CORRECT, WELL DONE”)
180 P$ = “”
190 GOTO 50
200 CALL SAY(“SORRY, YOU SPELL IT”)
210 FOR J = 1 TO LEN(X$(I))
220 CALL SAY(SEG$((X$(I),J,I))
230 NEXT J
240 P$ = “”
```

```
250 GOTO 50
260 DATA SMALL, PLEASE, PROBLEM, TURN, WEIGHT, WHICH,
    WHITE, POINT, YELLOW, SECOND
```

Conclusions

This book has aimed to provide an easy introduction to using the TI 99/4A microcomputer, and it has described many of the applications in which the TI 99/4A can be used to good effect simply by loading and running a program. There are a large number of applications of this kind, including some business applications, which require no knowledge of how the TI 99/4A microcomputer works and need only a minimal knowledge of the instructions required to operate it. In these circumstances the program is all important. The TI 99/4A microcomputer is merely a vehicle for running the program. A special purpose system of this kind can demonstrate its worth by paying for itself in quite a short time. However, the TI 99/4A microcomputer is extremely versatile, being capable of as many activities as it can be programmed for. This versatility can be harnessed by running different programs for each of a range of applications.

Purchased programs do not always do exactly what you may want, so it is useful to be able to program the TI 99/4A microcomputer in order to modify such programs. Whether for this reason, or as a result of inquisitiveness about how to tap the full potential of the TI 99/4A microcomputer, it is useful to be able to write programs. An introduction to programming the TI 99/4A microcomputer is provided by this book, but it is only an introduction and Appendix 1 indicates several sources of information which can be used for further study.

The importance of the TI 99/4A microcomputer used as an educational tool has been stressed more than once in these pages. Its importance as an example of modern technology should not be overlooked. It has a merit merely existing as an available product of the technology that will be used increasingly in the future, by providing an appreciation of how the technology is applied in everyday situations. In particular, the TI 99/4A's unique Speech Synthesizer is a good example of the early stages of advancing computer technology.

When viewed from different perspectives, the TI 99/4A microcomputer is seen as a tool which can be used in a number of ways. This book has attempted to introduce many of these uses and to indicate the sources of information which will help in developing these avenues further.

Appendix 1
Further reading

This appendix lists some books and magazines that are suitable for further reading to follow up particular topics that are mentioned, introduced or developed within the book. At the time of writing, we know of no other book specifically on the TI 99/4A computer in this country. This is probably because Texas Instruments has only recently started to actively market the computer outside of North America. If this is the case, we can expect to see many more books appearing in the very near future; both those written in the UK, and also imports of American books. There is, however, already a UK TI Home Computer Users' Club, at 157 Bishopsford Road, Morden, Surrey.

Magazines

Computing Today
This magazine is considered by many to be the best of the popular computing magazines. It covers the whole field of microcomputing and often provides listings of useful programs. Converting these to run on the TI 99/4A could be both entertaining and educational.

Which Micro and Software Review
This is one of the better of the new crop of computer magazines. It features articles on a wide range of equipment from the lower end of business machines to the new small home computers. Articles on the TI 99/4A do appear now and again, but it is particularly useful for gaining an overall view of the general trends in microcomputing.

Personal Computer World
Often abbreviated to *PCW*, this is in many ways required reading for microcomputer users.

Micro Software and Systems Magazine
This is a new magazine that is devoted mainly to business software. Each issue focuses on a different computing application: graphics, operating systems, word processing, etc. It may be of interest to those who intend to use their TI 99/4A to develop serious software, by providing an outline of the existing software in specific fields.

99'er Magazine
An independent magazine launched in the US for users of the TI 99/4A. It is published bi-monthly and features editorial comment and detailed articles on a wide range of subjects of interest to the TI user. For further details write to:

99'er Magazine
PO Box 5537
Eugene, OR 97405
USA

Magazine articles

Speech Synthesis with Linear Predictive Coding, by Larry Brantingham, Interface Age, June 1979, pp 72-75.

Electronics Speaks Out, by N C Pearson, Design News, April 9, 1979, pp 76-79.

Three-Chip System Synthesizes Human Speech, by Richard Wiggins and Larry Brantingham, Electronics, August 31, 1978, pp 109-116.

General books

Illustrating BASIC, by Donald Alcock (Cambridge University Press, 1978).
Somewhat dated now, but still the best general introduction to BASIC programming available. Like Dennis Jarrett's book (see below), an easy style can (wrongly, we think) be interpreted as a patronising approach from the author.

Software Secrets, by Graham Beech (Sigma Technical Press, 1981).
This book is actually written for a Sharp MZ-80K microcomputer, but since it is really a book about programming ideas and techniques, it makes useful and interesting reading.

The Good Computing Book for Beginners, by Dennis Jarrett (ECC Publications Ltd, 1980).
Claimed to be "all you need to know about computers (and nothing you don't)"; its main use is in an extensive glossary (over 220 pages!). Jarrett writes in an easy and colloquial style ("ECMA – It sounds like a skin complaint but it stands for the European Computer Manufacturers' Association"). If you object to the style, don't buy the book!

Inside BASIC Games, by R Mateosian (Sybex).
Teaches interactive BASIC programming through games.

BASIC Computer Programs for the Home, by Charles D Sternberg, Hayden, 1980.
A comprehensive book of practical home application programs that will be helpful to both the novice and experienced owner by increasing the usefulness of any home computer.

The First Book of Microcomputers, by Robert Moody, Hayden.
Claimed to be the home computer owner's 'best friend'.

The BASIC Workbook, Kenneth Schoman, Jr, Hayden.
A hands-on approach to learning BASIC and the fundamentals of problem-solving using a computer. The book is subtitled: 'Creative techniques for Beginning Programmers'.

Appendix 2
Glossary

Access
To obtain data from, or place data into, storage; which may be either main memory or backing storage.

Address
The storage location of information, either in the computer's memory, or on cassette tape or floppy disk.

Argument
Commonly used to describe the value associated with a command.

Array
A linear arrangement of individual items of data that can each be identified by an index that allows single items to be examined. Thus, the command DIM A$(20) will provide an array of 20 memory locations that can be examined sequentially by their names A$(1), A$(2) and so on.

ASCII characters
The American Standard Code for Information Interchange (pronounced 'as-key'): a code used by most computers to represent 128 different text and computer control characters. It uses 7 bits for each character. For example, the ASCII code for the character A is 1000001.

Assembly language
A language similar in structure to machine language, but made up of mnemonics and symbols. Programs written in assembly language are slightly less difficult to write and understand than programs in machine language.

BASIC
The computer language immediately available when many microcomputers are turned on (including the Dragon and the BBC Microcomputer), and in which commands to it are expressed. BASIC actually stands for Beginner's All-purpose Symbolic Instruction Code.

Binary
A number system with two digits, '0' and '1', with each digit in a binary number representing a power of two. Most digital computers are essentially binary in nature.

Bit
Short for 'binary digit'. A bit (0 or 1) is the smallest unit of digital information.

Board
A printed circuit board, or PCB, is sometimes called a printed circuit card. It is usually plastic and has its required circuits (in a conducting medium like copper) printed on its surface. There is also a number of small holes where individual electronic components can be plugged or soldered into the board to make contact with those circuits. At the other end of the circuit is the edge of the board, which is equipped with connectors. The connectors engage with further circuitry in the backplane, which is the part of the computer that interconnects the various boards. The processing functions of a microcomputer and its main memory will be held on a small number of PCBs.

Boot
Short for 'bootstrap': the process of loading an operating system from disk or tape into computer memory.

Branch
A departure from the sequential performance of program statements. An unconditional branch causes the computer to jump to a specified program line every time the branching statement is encountered. A conditional branch transfers program control in accordance with the result of a conditional test.

Buffer
An area of computer memory for temporary storage of either input or output data.

Bug
An error in computer programming. Because the error may only be noticed when it affects a different part of the program, many bugs are very difficult to find.

Byte
A unit of computer storage that comprises 8 bits. It is almost the same as the storage needed for a single character. Thus a 32K Byte computer has approximately 32000 storage locations each able to store a character.

Central processing unit (cpu)
The 'brains' of the computer, containing the electronic circuits that interpret and execute instructions.

Character
A letter, number, punctuation symbol, or special graphics symbol.

Chip
Literally, the chip of silicon from which an integrated circuit is fabricated, but used popularly to refer to the integrated circuit itself.

Constant
A specific numeric or string value. A numeric constant is any real number, such as 1.2 or −4321. A string constant is any combination of characters, up to the limit set by each different version of BASIC, enclosed in quotation marks, such as "HELLO" or "221 Baker Street". See also 'Variable'.

Cursor
The flashing bar or square on a computer's visual display screen which indicates the position at which the next item will be displayed.

Database
An organised collection of data from which either data or the properties of items of data can easily be retrieved.

Debug
To find and correct errors (bugs) in a program.

Default
This is a value that is automatically assigned by the program being used whenever the user of that program does not specify a particular value for a given variable.

Disk
A disk on which programs or data can be stored as magnetic patterns on the surface of the disk, and from which recorded information can be rapidly retrieved. Also known as a floppy disk.

Disk Operating System (DOS)
An operating system specifically for a disk drive. A program to facilitate the storage of information on disk and its retrieval from the disk. The DOS selects unused portions of the disk surface for data storage, and then remembers where everything is for data retrieval. See also, operating system.

Double-density
It is possible for disk drive manufacturers to double the number of bits stored per inch on a disk. You pay extra for the drives and for the disks themselves for this increased storage capacity – but the price increase is

less than that for buying a second disk drive. Since double-density drives came on to the market, the original density disks are now often referred to as 'single-density'.

Double-sided
It is now possible for disk drive manufacturers to produce disks and disk drives with data storage facilities on both sides of the disk. The extra technology involved in writing to and reading from both sides of a disk means that these disks are far more expensive than is usually acceptable for home computers. These drives are therefore more frequently found on business computers. Since double-sided drives came on to the market, the original disks are now often referred to as 'single-sided'.

Execute
The act of obeying the instructions contained in a computer program. Synonymous with running a program.

File
A collection of related data records stored on a device, such as a cassette tape or floppy disk.

Floppy disk drive
A peripheral device used to store programs and data on disks made of a thin flexible plastic coated with a magnetic recording surface (called a floppy disk or diskette). Floppy disks are more reliable and much faster in operation than simple cassette tapes.

Flow chart
A diagram indicating in stylised form the steps of a computation. It is used as an aid to program development.

Graphics
Pictures produced by a computer.

Hardware
More properly called 'computer hardware', it is the collection of physical devices that make up a computer system.

High level language
A language that is more intelligible to human beings than it is to machines; for example, BASIC, Pascal, FORTRAN. See also: Low level language.

Increment
A value that consistently modifies a variable. The FOR . . . NEXT . . . STEP instruction consistently modifies (increments) the FOR variable by the STEP value.

Integer
A whole number, either positive, negative, or zero.

Integrated circuit
An electronic circuit fabricated in extreme miniature form on a silicon chip typically a few millimetres square.

Interface
An electronic and/or physical connection between different devices. A serial interface transmits or accepts information one bit at a time, whereas a parallel interface transmits or accepts information several bits at a time.

Interpreter
Software which translates a program in a high-level language into machine code, which comprises the binary instructions which correspond directly to computer operations and is the 'language' that the microprocessor understands. The program is executed at the same time as it is interpreted. This is distinct from a 'compiler', which performs a similar operation but produces a compiled program from the user's source program. This compiled program is the program that is ultimately executed. With an interpreter, each statement in the high-level language program is translated and executed immediately. This means you can add or delete instructions and see the effect immediately, so it speeds the process of program development. Interpreters might take up some memory, since they have to be waiting to translate; and interpreted programs are certainly slower when it comes to run-time (because a program already in machine code is inevitably much more efficient). But because interpreter languages do not require the compile process they are generally preferred for home computers. Apart from BASIC, you will find the APL and PASCAL languages frequently in interpreter form.

K
1K stands for 1 kilobyte of memory, and gives the size of memory consisting of multiples of 1024 storage locations.

Listing
A printout (which can be either as a display on the screen, or as a physical printed list) of the lines of instruction that make up a program.

Loop
A group of consecutive program lines that are repeatedly performed, usually a specified number of times.

Low level language
A language that is more intelligible to machines than it is to human beings; for example, assembler. See also: High level language.

Machine code
The code in which instructions must be conveyed to a microprocessor in order that it may respond to them directly.

Memory
Also called main memory, core memory, or main storage. The integrated circuits of a computer in which information is stored that is directly accessible to the cpu, as opposed to peripheral, or backing storage which is accessible only via interfaces.

Microcomputer
A computer whose central processor is on a microprocessor.

Microprocessor
Physically, a very complex integrated circuit. Functionally, an electronic device that can be programmed and can, in consequence, perform a variety of tasks.

Mode
A condition or a set of conditions under which a particular set of rules applies.

Operating system
Systems software that controls the computer and its peripheral devices.

Output
Information sent by the cpu to any peripheral device.

Peripheral
Equipment that can be attached to a computer, and can be used and controlled by the computer. Examples are cassette units, television screens, and printers.

Port
A socket on the computer into which you can plug a terminal or some other input/output device.

Printed circuit board
(see Board)

Program
An ordered sequence of commands given to a computer, so that when it obeys them it automatically performs a specified task.

Prompt
A symbol (different for the different versions of BASIC) which marks the beginning of each program line during input from the keyboard; a symbol or phrase that requests input from the user.

RAM
Random-access memory. Memory whose contents is lost when the power supply is turned off. The amount of RAM determines how much memory is available for the user to store programs and data.

Record
A collection of related data elements, such as an individual's payroll information or a student's exam scores. A group of similar records, such as a company's payroll information, or a school's exam results, is called a file.

ROM
Read-only memory. This is permanent memory, typically used to store information that is always required, such as that which provides BASIC. This memory is not available to store the user's programs: it provides facilities required by the user.

Scroll
To move all the text on the screen of a video monitor (usually upwards) in order to make room for more (usually at the bottom).

Software
Computer programs; the list of instructions that tell a computer to perform a given task or tasks – as opposed to hardware (the computer itself). There are basically two types of software: systems and applications. Examples of systems software include operating systems and language interpreters. Applications software includes programs that instruct the computer to perform specific applications, such as word processing, computer games etcetera.

String
A sequence of letters, numbers or symbols, usually arranged in some specific order, and treated as a unit.

Subroutine
A program segment which can be used more than once during the execution of a program, such as a complex set of calculations. In most forms of BASIC, a subroutine is best defined with the GOSUB and RETURN statements.

Trace
Listing the order in which the computer performs program statements. Tracing the line numbers can help you find errors in a program flow.

User
Any person or persons who use a computer.

User port
One of the connections at the rear of a number of microcomputers, which can be used to send or receive signals under the control of the user's program.

Users group
A group of people who have computers from a particular supplier, or who have some kind of common computing interest. It is worth stressing the value that this kind of organisation can offer, even when, as sometimes happens, it is really a manufacturer-inspired mouthpiece designed primarily for marketing purposes. Users can discuss problems, swap solutions and programs, band together to get discounts on bulk-buying consumables like paper and disks, and if necessary present a coherent front to get some action from the supplier.

Variable
A name given to a value that may vary during the execution of a program. Think of a variable as a memory location or pigeon-hole where values can be replaced by new values during program execution.

Word processor
A system for processing textual material electronically and then printing it or, perhaps, transmitting it to a similar system. In this context, the processing is mainly editing.

Appendix 3
The Tower of Hanoi – a Game

This game is one of the classic ancient logic problems. You are given a pile of discs of different sizes, and you have to move the discs one at a time from location A to location C in as few moves as possible. Location B is available as a temporary position to help you sort the discs. The problem, however, is that you must never put a disc on top of one that is smaller in size than itself.

To use the program, enter the following listing accurately. When entered, type RUN and follow the instructions that appear on the screen. You will soon be presented with a picture showing a Tower of the size you chose on the left hand side. Locations B and C, empty to begin with, are shown to the right. To move the discs, simply type the location of the disc you wish to move (say, 'A') followed by the location you wish to move it to (say, 'C'). Thus, 'A,C' may well be your first move. If it is, then 'A,B' must be your second move – you cannot do 'A,C' again (big disc on to little disc!), and there is little point in moving the small disc again before making any other move. Good luck!

```
 10 DIM A(10,5),V(30),W(15)
 20 CALL CLEAR
 30 CALL SCREEN(8)
 40 PRINT TAB(10); "TOWER OF HANOI"
 50 PRINT
 60 PRINT
 70 PRINT "AIM: MOVE ALL DISCS TO C"
 80 PRINT
 90 PRINT "RULE 1: ONLY MOVE ONE DISC EACH MOVE"
100 PRINT
110 PRINT "RULE 2: LARGE DISCS MUST NOT SIT ON SMALL
    DISCS"
120 PRINT
130 PRINT "PRESS ANY KEY TO START"
140 PRINT
150 CALL KEY(3,A1,B)
```

```
160 IF B = 0 THEN 150
170 CALL CLEAR
180 PRINT
190 PRINT TAB(10); "TOWER OF HANOI"
200 PRINT
210 PRINT "HOW MANY DISCS DO YOU WANT?"
220 PRINT
230 PRINT "CHOOSE A NUMBER BETWEEN 2 AND 5"
240 PRINT
250 PRINT
260 CALL KEY(3,A1,B)
270 IF B = 0 THEN 260
280 IF A1 > 49 THEN 300
290 GOTO 310
300 IF A1 < 54 THEN 360
310 PRINT
320 PRINT "YOU HAVE TYPED AN INVALID NUMBER"
330 FOR I = 1 TO 500
340 NEXT I
350 GOTO 170
360 C = A1-48
370 FOR I = 1 TO 21
380 READ V(I)
390 NEXT I
400 DATA 73,78,86,65,76,73,68,32,77,79,86,69
410 DATA 84,82,89,32,65,71,65,73,78
420 FOR I = 1 TO 11
430 READ W(I)
440 NEXT I
450 DATA 84,89,80,69,32,65,32,77,79,86,69
460 FOR I = C TO 1 STEP -1
470 READ A(I,1)
480 A(I,2) = 0
490 A(I,3) = 0
500 NEXT I
510 DATA 1,2,3,4,5
520 H(1) = C
530 H(2) = 0
540 H(3) = 0
550 CALL CLEAR
560 CALL SCREEN(7)
570 PRINT
```

```
580 PRINT TAB(9); “TOWER OF HANOI”
590 PRINT
600 PRINT
610 PRINT “TARGET MOVES(SPC)”;2∧C−1
620 PRINT
630 PRINT “MOVES MADE(SPC)”;M
640 FOR I = 1 TO 17
650 PRINT
660 NEXT I
670 A$ = “FFFFFFFFFFFFFFFF”
680 B$ = “0000000000000000”
690 CALL CHAR(40,A$)
700 CALL CHAR(41,B$)
710 CALL CHAR(96,A$)
720 CALL COLOR(9,14,7)
730 CALL HCHAR(19,1,96,32)
740 CALL HCHAR(20,1,96,32)
750 CALL HCHAR(21,1,96,32)
760 CALL HCHAR(22,1,96,32)
770 CALL HCHAR(20,5,65)
780 CALL HCHAR(20,15,66)
790 CALL HCHAR(20,25,67)
800 GOSUB 5000
810 GOSUB 6000
820 IF H(I1) > 0 THEN 940
830 FOR I = 1 TO 12
840 CALL HCHAR(6,18+I,V(I))
850 NEXT I
860 FOR I = 13 TO 21
870 CALL HCHAR(8,6+I,V(I))
880 NEXT I
890 FOR I = 1 TO 500
900 NEXT I
910 CALL HCHAR(6,19,41,12)
920 CALL HCHAR(8,19,41,9)
930 GOTO 800
940 IF H(I2) = 0 THEN 960
950 IF A(H(I1),I1)>A(H(I2),I2) THEN 830
960 H(I2) = H(I2)+1
970 A(H(I2),I2) = A(H(I1),I1)
980 A(H(I1),I1)=0
990 H(I1) = H(I1)−1
```

```
1000 M = M+1
1010 M$ = STR$(M)
1020 FOR I = 1 TO LEN(M$)
1030 CALL HCHAR(6,14+I,ASC(SEG$(M$,I,1)))
1040 NEXT I
1050 GOSUB 5000
1060 IF H(3) <> C THEN 800
1070 PRINT “CONGRATULATIONS!!!!”
1080 GOTO 1080
5000 CALL COLOR(2,11,7)
5010 FOR P = 1 TO 3
5020 FOR I = 1 TO C
5030 R1 = 1+10★(P−1)
5040 R2 = 5−A(I,P)
5050 R3 = 2★A(I,P)−1
5060 R4 = 20 − I★2
5070 IF A(I,P) = 0 THEN 5110
5080 CALL HCHAR(R4,R1+R2,40,R3)
5090 CALL HCHAR(R4−1,R1+R2,40,R3)
5100 GOTO 5130
5110 CALL HCHAR(R4,R1,41,9)
5120 CALL HCHAR(R4−1,R1,41,9)
5130 NEXT I
5140 NEXT P
5150 RETURN
6000 CALL COLOR(5,2,7)
6010 CALL COLOR(6,2,7)
6020 CALL COLOR(7,2,7)
6030 CALL COLOR(8,2,7)
6040 FOR Y = 1 TO 11
6050 CALL HCHAR(6,18+Y,W(Y))
6060 NEXT Y
6070 CALL KEY(3,A1,B)
6080 IF B = 0 THEN 6070
6090 FOR Y = 1 TO 11
6100 CALL HCHAR(6,18+Y,41)
6110 NEXT Y
6120 CALL KEY(3,A2,B)
6130 IF B = 0 THEN 6120
6140 I1 = A1−64
6150 I2 = A2−64
6160 RETURN
```

Index

Kevin Townsend

LEARNING TO USE THE TI99/4A COMPUTER

This beginners' guide really does begin at the beginning. It assumes that you want to learn to *use* the TI99/4A computer in your work or leisure, not become a theorist in computing. *Learning to Use the TI99/4A Computer* provides a simple, down to earth, jargon-free introduction to the machine and its software. Follow the text and illustrations and you will end up operating the TI99/4A and understanding its many capabilities.

Many applications of the TI99/4A are described, including business, educational and hobby uses. Additionally, a simple and direct introduction to programming the TI99/4A is given in a way which will help motivate the user to further investigation of the TI99/4A's capabilities. The TI99/4A's ability to produce and draw pictures and diagrams is explored and explained, and programs for a large number of graphics applications are presented.

This book will appeal to new TI99/4A owners, students in schools and colleges where TI99/4As are used, businessmen who wish to learn about how to use the TI99/4A and program it. It will help those who are already learning to use the TI99/4A, but find their current manuals difficult to follow. It also provides the would-be purchaser of microcomputers with information on how the TI99/4A operates and performs, which will help him to assess whether the machine will suit his need.

About the series

This series of books has been designed to provide potential users, established users, teachers, students and businessmen with standardised introductions to the use of popular microcomputers. Extensive use has been made of photographs, diagrams and drawings to illustrate the text and make it easy to read and understand.

As the layout and content of the books in the series are similar, each book may be used in conjunction with others for purposes of comparison of performance and capabilities. The *Learning to Use* series is an inexpensive way of checking that the would-be purchasers' provisional choice of machine is the correct one.

The series is open-ended and will cover new models of microcomputers as they appear on the market.

Titles in the series

Learning to Use the PET Computer
Learning to Use the ZX81 Computer
Learning to Use the BBC Microcomputer
Learning to Use the VIC-20 Computer
Learning to Use the ZX Spectrum Computer
Learning to Use the Dragon 32 Computer
Learning to Use the Commodore 64 Computer
Learning to Use the TI99/4A Computer
Learning to Use the Oric 1 Computer
Learning to Use the Apple II/IIe Computer
Learning to Use the Lynx Computer

£4.95

Gower

ISBN 0 566 03492 1
Gower Publishing Company Limited, Gower House, Croft Road,
Aldershot, Hampshire GU11 3HR, England